Securing Communication Channels in IoT using an Android Smart Phone

Master of Science (Tech.) Thesis
University of Turku
Department of Future Technologies
Security of Networked Systems
2020
Amel Bourdoucen

Supervisors:
Seppo Virtanen, PhD
University of Turku
Philip Ginzboorg, PhD
Huawei Technologies Oy
Antti Hakkala, PhD
University of Turku

UNIVERSITY OF TURKU
Department of Future Technologies

Amel Bourdoucen: Securing Communication Channels in IoT using an Android Smart Phone

Master of Science (Tech.) Thesis, 70 p.
Security of Networked Systems
June 2020

In today's world, smart devices are a necessity to have, and represent an essential tool for performing daily activities. With this comes the need to secure the communication between the IoT devices in the consumer's home, to prevent attacks that may jeopardize the confidentiality and integrity of communication between the IoT devices.

The life cycle of a a simple device includes a series of stages that the device undergoes: from construction and production to decommissioning. In this thesis, the Manufacturing, Bootstrapping and Factory Reset parts of IoT device's life cycle are considered, focusing on security. For example, the Controller of user's home network (e.g., user's smart phone) should bootstrap the "right" IoT device and the IoT device should bootstrap with the "right" Controller. The security is based on device credentials, such as the device certificate during the bootstrapping process, and the operational credentials that are provisioned to the IoT device from the Controller during the bootstrapping.

The goal of this thesis is to achieve easy-to-use and secure procedure for setting up the IoT device into a home network, and for controlling that IoT device from an Android mobile phone (Controller). The objectives are: (1) explore the different aspects of using a smartphone as a Controller device to securely manage the life cycle of a simple device; (2) propose a system design for securely managing the life cycle of a simple device from a Controller compliant with existing standards, (e.g. Lightweight Machine to Machine (LwM2M) is an industrial standard used to manage and control industrial IoT Devices); (3) implement a proof of concept based on the system design; (4) provide a user-friendly interface for a better experience for the user by using popular bootsrapping methods such as QR code scanning; (5) discuss the choices regarding securing credentials and managing data, and achieve a good balance between usability and security during the bootstrapping process.

In order to achieve those goals, the state-of-art technologies for IoT device management were studied. Then an Android application that uses LwM2M standard in consumer's home setting was specified, designed and implemented. The Android application is wrapped in a smooth user interface that allows the user a good experience when attempting to connect and control the target IoT device.

Keywords: IoT Security, Out-of-Band, Authentication, QR Code.

Glossary - Abbreviations

AKE	Authenticated Key Exchange
CoAP	Constrained Application Protocol
DoS	Denial of Service
DTLS	Datagram Transport Layer Security
EAP	Extensible Authentication Protocol
EC	Elliptic Curves
GUI	Graphical User Interface
HAP	Homekit Acessory Protcol
HTTP	Hypertext Transfer Protocol
IC	Integrated Circuit
IoT	Internet of Things
JCA	Java Connector Architecture
JWT	JSON Web Tokens
LwM2M	Light Weight Machine to Machine
MAC	Message Authentication Code
MFi	Made for iPhone/iPod/iPad
MitM	Man in the Middle
NFC	Near Field Communication
OCR	Optical Character Recognition
OOB	Out-Of-Band
PK	Public Key
PSK	Pre-Shared Key
QR	Quick Response code
REST	Representation State Transfer

SRP	Secure Remote Protocol
SSID	Service Set Identifier
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TEE	Trusted Execution Environment
TLS	Transport Layer Security
TPM	Trusted Platform Module
TTP	Trusted Third Party
UDP	User Datagram Protocol

Terminology

Bootstrapping	Bootstrapping is the process of changing a device from not operational to operational modes.
Controller	Typically a sophisticated device, e.g., smart phone that is authorized to connect and send commands to a simple device.
Factory reset	Restores a simple device to its original manufacturer's settings, by erasing the configurations and data that have been accumulated by the device during its usage.
In-band communication	Occurs over the same physical communication channel (e.g., the home WiFi network) that carries the bulk of data between two devices.
Out Of Band (OOB) Communication	Happens over a physical channel that is different from the in-band channel. OOB communication typically involves the user.
Simple Device	A resource constrained device that is connected to a network where it can receive commands from a Controller device and act accordingly.
Management Data	Data that is stored in the device in operational mode. These datas are necessary during the operational mode of the device which includes configuration parameters, contextual information and operational credentials.
Nodes	When an IoT device is part of an IoT network it is called a node.

Contents

1 Introduction

The term Internet of Things (IoT) was first mentioned in 1999 by Kevin Ashton exclusively for supply chain management. Today, the concept of IoT includes a wide range of applications such as transportation and healthcare [1]. Simple devices are resource-constrained, physical IoT devices.

IoT devices range from simple, resource-constrained devices, like light switches, to more complex devices, such as surveillance cameras. Resource-constrained, simple IoT devices require specialized protocols that accommodate to their restricted requirements such as limited energy or RAM. Two examples of protocol suites designed for IoT devices are 6LoPan [2] and the Zigbee [3]. The Constrained Application Protocol (CoAP) is a specialized transfer protocol that enables simple devices to interact with other nodes in an IoT network with low bandwidth. CoAP can run of different transmission protocols, such as SMS or Bluetooth.

1.1 Problem Definition and Goal

Secure management and operation of smart devices in people's homes is one of the key issues in consumer IoT. This thesis investigates an approach to solving this issue by adapting the LwM2M industrial standard (LwM2M [4]) and using the smart phone as the Controller of home IoT devices. In the home setting, where the user needs to be involved during the bootstrapping (initial setup) of the simple device by the Controller, the bootstrapping process should be user-friendly, while

still maintaining the required level of security. In the current LwM2M specification the bootsrapping entity in the Controller has to be pre-configured in the IoT device before the bootstrapping operation. Thus, the LwM2M bootsrapping needs to be adapted to the consumer's home setting.

The objectives of this master's thesis are to: (1) explore and investigate different aspects of using a smart phone as a Controller device to securely manage the life cycle of a simple IoT device; (2) Propose a system design for securely managing the life cycle of a simple device from a Controller compliant with LwM2M standards; (3) implement the design in a context of a consumer solution. The security requirements in this context is of data confidentiality and proximity authentication using an application that supports LwM2M protocol and exchanges messages using CoAP over DTLS; (4) to study the usability aspect of the design/implementation and (5) discuss and analyse major findings regarding securing credentials and managing data and secure usability that requires users intervening in the bootstrapping process. Home IoT is the network of devices that this design solution creates and *HomeApp* is the application that is running on the phone (Controller) which represents the command and control unit. The controller uses DTLS protocol to secure CoAPs messages which resides in the application layer. The proximity authentication is due to the IoT device being in the range of a mobile phone.

1.2 Research Questions

Based on the problem description and the objectives of this thesis, the following research questions are investigated:

- RQ1: How to securely manage the life cycle of the device using a Controller device?

- RQ2: What types of data should be transferred between the Controller and

the simple device and how to securely transfer and store them safely during bootstrapping (initial setup) and operational modes (after bootstrapping)?

- RQ3: How can the Controller protect long-term credentials that need to be provisioned to the simple device against cyber attacks?
- RQ4: How to provide a user friendly experience during the bootstrapping of the device using QR code reading?

1.3 Structure of thesis

This document is structured to include six chapters. Chapter 2 introduces a background reading that highlights the key concepts used in the thesis, in addition to an illustration of the current technologies that use a similar approach to problem solving of this topic. Chapter 3, is a description of the structure of the system architecture and system components that include the LwM2M Client, LwM2M Server, the different interfaces and entities. Chapter 4 describes the implementation that previews the technical solutions used to solve the problem. Chapter 5 aims at discussing and evaluating the solution discussed in chapter 4. Chapter 6 is the conclusion, which summarizes the results and benefits obtained from this work and possible improvements in the future.

2 Literature Review

2.1 Introduction to Life cycle Management

Life cycle management is a series of stages that the device undergoes from inception to decline. The stages include: (1) construction and production of a device by the manufacturer. (2) securely setting up and (3) managing of the device using a controller such as a smart phone. Finally, (4) decommissioning which includes factory reset and the end of life for a device [5]. The same paper by Rahman *et al.* [5] contributes with a generic life cycle model that describes the stages that an IoT device under goes from (re)construction to decommission. The generic life cycle model is in the Figure 2.1 adapted from [5].

The first stage (1) is the production stage where the devices are produced in a large scale including the installation of the necessary software. (2) The deployment and commission phase prepares the device for operation and consists of necessary configurations. In this phase the Controller pairs with the device using available/-supported pairing methods. In addition to these configurations, the Controller acquires the necessary secret keys for authentication and secure communication within the network. This is also known as Bootstrapping. (3) The operation stage, consists of execution of services and applications i.e, switching on and off a coffee machine. (4) The update stage, where management functions are performed to update the device. Finally, the (5) decommissioning stage where the device reaches its end of

life. Here the device is disconnected from the network and no longer accessible.

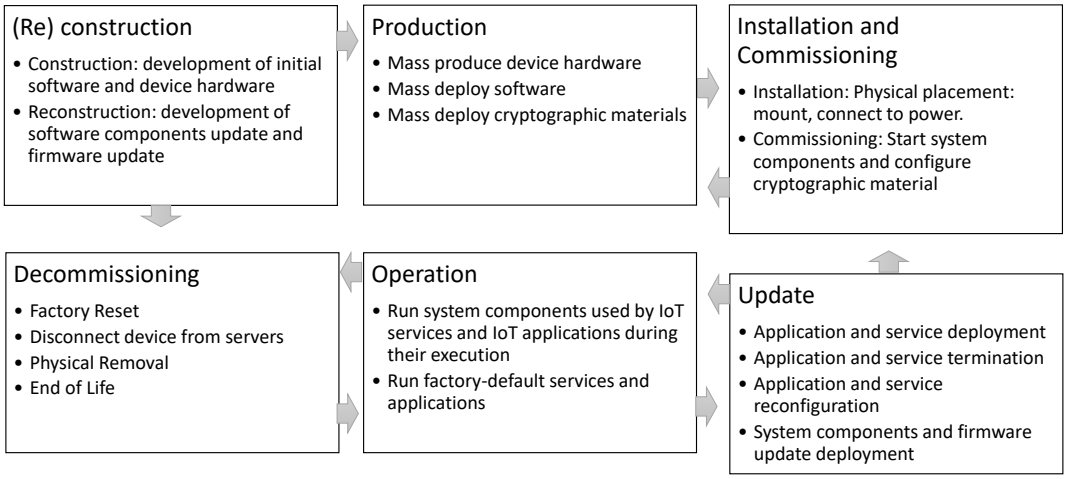

Figure 2.1: Generic Life cycle for IoT Devices (Adapted from [5])

The primary contribution of this thesis is to identify the role of the Controller in securing the life cycle management operations of a simple device. The Bootstrapping phase is critical in this aspect as it is the phase where the device gets its credentials that are used to secure the operational stage operations. Therefore, it is important to explore and understand various security aspects of bootstrapping.

2.2 Bootstrapping in IoT

Bootstrapping is defined as the procedure in which the user attains parameters and other security configurations that represent the authentication and authorization means by which the user is verified into the security domain at a certain time and location [6] [7]. Bootstrapping can occur in different stages in a life cycle of a device and can include provisioning details that have been implemented previously during

the manufacturing phase. It is vital to focus in this thesis on bootstrapping of related security information since once a channel is secured between a simple device and a Controller for communication, the remaining information can be shared easily. Bootstrapping is also important as it establishes a basis of trust anchor which shall be used to secure all communication henceforth between the Controller and the simple device. Bootstrapping is the foundation of securing simple device life cycle management. In terms of the classification of the available mechanisms of Bootstrapping, some Bootstrapping tools are more user-intensive while others require less involvement from the user. An example for this can be a user scanning a QR code from the Controller to initiate the pairing process with a device. According to Garcia-Morchon *et al.* [6] there are several classifications of Bootstrapping methods:

- Managed methods: pre-established trust relationship is used alongside authentication credentials. The process takes place through authentication by centralized servers. An example of this is Extensible Authentication Protocol (EAP) [8] which was designed to work during the network authorization phase. It is applicable when the IP layer connectivity may not be present for use. EAP is a datalink layer protocol and doesn't support message rendering/handling; meaning that EAP cannot transmit bulk data unlike TCP or SCTP. Another difference between EAP and other protocols is that in the case of EAP, the authentication is initiated from the server instead of the client.

- Peer-to-Peer and Ad-hoc methods: these methods do not rely on pre-established trust. The credentials are built as a result of subsequent secure communication. The procedure includes unauthorized Diffie-Hellman exchange (secure key exchange method) followed by an out-of-band (OOB) communication channel to avoid man-in-the-middle (MitM) attack.

- Opportunistic and leap-of-faith methods: Unlike other methods, this method

does not verify the initial authentication, instead it verifies the continuity of the connecting; assuming the attacker is not present during the initial step. An example of this is Wifi Protected Setup (WPS) and Secure Shell (SSH) [9].

- Hybrid methods: use components from both the ad-hoc and managed methods.

To provide authenticity in bootstrapping, Stajano and Anderson [10] proposed a metaphor called *imprinting*. The metaphor uses a naturally occurring process called *imprinting* in animals and specifically ducklings. As the duckling emerges from its egg it will recognize the first object it sees (that moves and makes a sound) as its mother. The device is viewed to be the body of the duckling while the state is said to be the software. The alive state of the duckling depends on the software being present in the body and it is bound to the mother that it has imprinted. Upon death, the device is returned to a pre-birth state where it is ready to accept new imprinting. This is also known as *reverse metempsychosis* [11]. During the imprinting phase, a secret shared key is exchanged between the duckling and its mother. This raises the question of how many nodes are able to generate a shared secret and how can we be sure that the key has been established with the right device? The answer to this question is through physical contact. It is crucial at this point to differentiate between *interacting* with other entities than the mother and being *controlled* by them. In this case, a suggested solution is to always bootstrap by establishing a shared secret to gain more specific policies to the node.

This is the main reason why many internet protocols rely heavily on Authenticated Key Exchange (AKE) for setting up secure channels. For example: protocols such as TLS and TTP (Trusted Third Party). On the other hand, a risk stands in involving third parties such as humans in the AKE procedure; in the sense that many shortcomings can derive as a result, and lead to vulnerabilities in the AKE scheme [12]. This has the lead to appointing the user to be the third party in the authentication process. The users are involved with the help of OOB channels over

the actual network communication. The question now lies in the challenges to the security of pairing these devices based on human-interactive security protocols. A massive strength point in OOB is authentication and message integrity opposed to secrecy [13]. The same paper by Kainda, Flechais, and Roscoe [13] argues that users represent the weakest link as they are prone to distraction and have different levels of motivation regarding performing security tasks, however, it is important to ensure that the users are diligent to carry out their roles in the best way possible to avoid security failures.

2.2.1 OOB Communication

With the user involvement over OOB, many OOB channels exist to provide efficient mechanisms for bootstrapping security in ad-hoc IoT deployments as summarized in Table 2.1. Their benefits can outweigh their drawbacks and vice versa, and they vary in the level of user involvement in the bootstrapping process. Nowadays, quite few smart home devices (consumer IoT devices) push the need for a companion device such as a mobile phone to aid in the OOB process and/or later be used to control the smart device.

- **Manual Typing** Manual authentication techniques have been produced to allow wireless devices to authenticate each other through an insecure wireless channel with the help of manual transfer of data between the devices. The devices in the context proposed by Gehrman *et al.* have access to wireless communication channel, are under the control of a single human user and if they do not have a keypad they must at least have a single means of inputting data. An example of this process is typing the Wi-Fi password to join a home network. Several issues arise with this approach: (1) Users are more prone to making mistakes as the number of digits that needs to be typed grows in number. (2) Users might find this non-trivial step too demanding. A possible

solution which may have a serious security weakness, is to truncate the hash-code, for instance the first 16 or 32 bits which means 4 or 8 hexadecimal digits [14].

- **Visual** The most popular and widely used is the visual OOB, which is the QR Code that is used to encode information to the QR Code image. Initially, QR codes were used to track automotive parts in manufacturing plants [15]. Their use has exploded into different areas such as: (1) Advertising (such as encoding URLs into statues, geo-locations and posters for ease of access of more information), (2) Mobile payments, (3) Access Control Krombholz *et al.* [16] in their work proposed how a QR code can be used along side other security techniques. An application generates a QR code with an encrypted password encoded and is scanned by the PC's camera. Another example for using visual authentication is ChromeCast[1].(4) Augmented Reality and Navigation.

  Some ways in which QR codes can be altered and involved in different attack scenarios is by replacing the entire QR code with a malicious link encoded or modifying the individual modules of a QR code [15].

- **Audio** Devices that are communicating over a wireless network, and are present within an appropriate distance from each other can use audio channels to secure pairing of these devices with human assistance, a system called Loud-and-Clear (L&C) was developed based on this approach by Goodrich, Sirivianos *et al.* [17]. Another example to using audio in OOB channels is through using Sound-proof mechanism which relies on two-factor authentication with the second factor being the proximity of the user's mobile device to the device used to authenticate. The proximity of the two devices is checked by comparing the ambient noise recorded by their microphones [18]. Audio

[1]Chromecast: support.google.com/chromecast/.

based authentication is used in Amazon Alexa[2] and Apple Speaker[3].

- **NFC Radio** Near Field Communication (NFC) allows sharing of data between two devices that are within a certain proximity of about 3-5 inches [19]. Widely known vulnerabilities such as eavesdropping, data corruption and data modification exists on NFC technology.

[2]Amazon Alexa: developer.amazon.com/en-US/alexa/alexa-voice-service/auth

[3]Apple Speaker: support.apple.com/en-us/HT208241

OOB Channel Type	**Controller/ Phone Interface**	**Usability Aspects**	**Notes**	**References**
Manual Typing	Real or Virtual Keyboard	Hard to type correctly a long sequence of characters.	Example: user types WiFi password to join home network. (1) this usability difficulty lead to schemes where the user has to input/type only short strings: - Password Authenticated Key Exchange (PAKE); Manual Authentication (MANA) by Gehrmann et al. (2004). (2) Broadcom's "pushbutton": the long-term key is agreed-on using plaintext messages over the WiFi in-band channel. (3) Chromecast: during device setup the TV and the smart phone display a number. The user has to confirm (in the smart phone) that both displays show the same number.	[14]
Visual	Camera	QR may fade with time.	Reading a QR code seems to be a frequently used method in setting up today's devices. One possible reason for this is that people are proficient in using their smart phone cameras.	[15] [16]
Audio	Microphone	Audio setup may fail due to ambient noise.	Used in Amazon Alexa and Apple Speaker.	[18] [17]
NFC Radio	NFC Reader	The NFC tag is often hard to locate for the end user. People are often confused where they need to place the NFC reader.	Vulnerabilities include: Eavesdropping, Data Corruption and Data Modification.	[19]

Table 2.1: OOB channel types and their usages

2.3 CoAP - Constrained Application Protocol

As web services has become more popular and the dependency on fundamental Representation State Transfer (REST) has grown to be more demanding. The importance of Constrained RESTful Environment (Core) is directed at realizing the REST architecture in proper form for Constrained nodes. An example for this is 6LoWPAN that supports IPv6 packets in small linklayer frames, that has a direct impact on the delivery success probability for these packets. CoAP has been designed to serve the special requirements for the constrained environment. These constrains can range from building automation, energy and other Machine-to-Machine (M2M) applications. The role of CoAP is not to blindly compress HTTP, but instead realize a subset of REST common with HTTP which is optimized for M2M [20].

CoAP is based on UDP using asynchoronous messages unlike HTTPS where the messages are synchronous. There are four message types in CoAP: (1) Confirmable, (2) Non-Confirmable, (3) Acknowledgment and finally (4) Reset. *Confirmable* messages where a recipient sends an acknowledgment message to the sender. *Non-Confirmable* messages are ones that do not require acknowledgment, in the scenario where the messages are repeated for specific applications of an application. *Acknowledgment* messages require that a specific *confirmable* message has reached. While *Reset* indicates that a message has been received but the context is missing so it is unable to be processed. CoAP also used methods such as GET, POST, PUT AND DELETE quite similar to HTTP methods.

2.4 DTLS - Datagram Transport Layer Security

Some applications such as M2M require the use of fast delivery instead of reliable delivery, to avoid delay in delivery in re-transmission and reordering packet loss is acceptable. This calls for the use of UDP. DTLS uses UDP and the DTLS full

handshake is displayed in the Figure 2.2 from [21]. DTLS protocol is unreliable, and it starts with a *ClientHello* from the client to the server. The packet is re-transmitted when dropped which is likely since it uses UDP. Upon successful transmission, the client expects to recieve a *HelloVerifyRequest* from the server, which is important to avoid Denial of Service (DoS) attacks. If the client does not recieve this message, it could be that the *ClientHello* or the *HelloVerifyRequest* were dropped so they are sent again. The timer is recommended to be between 500 and 100 ms [21].

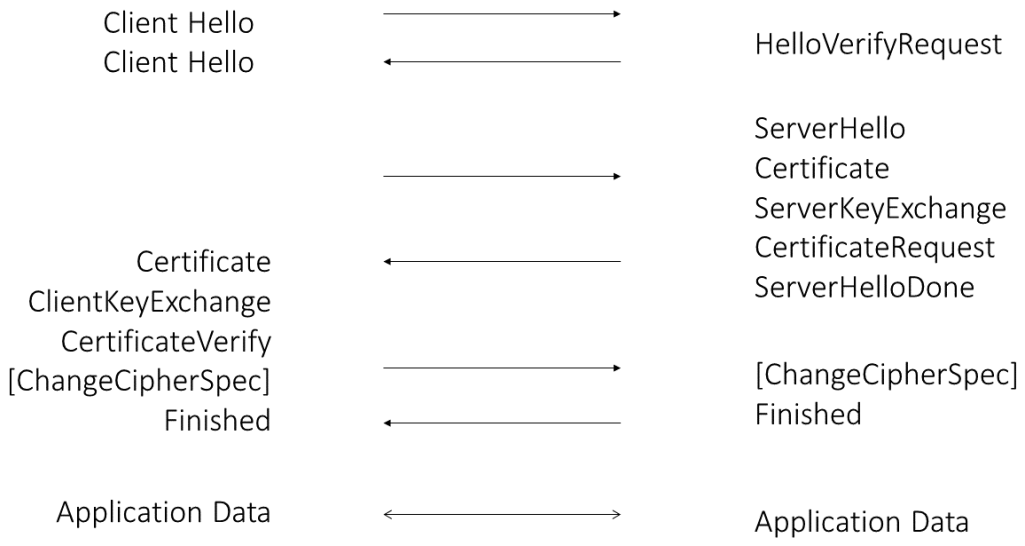

Figure 2.2: DTLS Handshake Messages [21]

2.5 Industry Solutions

In this section some of the recent technologies are summarized, in addition to the security measures that are taken during data exchange. The technologies discussed are: LwM2M by OMA, Apple Homekit, Xiaomi IoT Cloud and Google Cloud.

2.5.1 LwM2M

OMA SpecWork's LightWeightM2M (LwM2M) is a device management protocol. It was first designed to be used in sensor networks and to cater to the demands of M2M environment. More than 25 companies are using LwM2M in their products and services, for instance Nokia's IMPACT IoT Platform and Huawei's OceanConnect IoT Platform [4]. The protocol which is designed for remote management of M2M devices is based on REST architecture and builds on the secure data transfer standard Constrained Application Protocol (CoAP) [20]. The protocol supports various credentials for secure communications such as: certificates, raw public keys and pre-shared secrets which are mentioned for their use with TLS/DTLS [22].

According to OMA's technical specification [4], the overall structure of the LwM2M has four interfaces: (1) Bootstrap, (2) Client Registration, (3) Device Management and Service Enablement and finally (4) Information Reporting. It is required from OMA that all the communication between LwM2M clients, LwM2M servers and LwM2M Bootstrapping servers to have mutual authentication and that the communication to be encrypted and integrity protected between these entities (LwM2M clients, LwM2M servers and LwM2M Bootstrapping servers). When it comes to Ciphersuits, a set of algorithms are used to secure a network connection that uses SSL or even TLS. LwM2M Clients and LwM2M server support a limited number of CipherSuits that depend mainly on the following: their suitability for IoT devices, security reasons and their ability to improve interoperability. LwM2M supports *Elliptic Curves (EC)*, for ECDHE the group secp256r1 shall be supported. As well as, for ECDSA the curve secp256r1 shall be supported. It is important to note that the EC of less than 255 bits shall not be supported.

2.5.1.1 Bootstrap Modes

LwM2M defines four Bootstrapping modes: (1) Factory Bootstrap, (2) Bootstrap from Smartcard, (3) Client Initiated Bootstrap and (4) Server Initiated Bootstrap. In both the Client Initiated Bootstrap and the Server Initiated Bootstrap, they require the help of a LwM2M Bootstrap-Server to achieve the goal of connecting an LwM2M Client to their LwM2M Server(s).

(1) Factory Bootstrap In this mode, the LwM2M Client is configured with the required bootstrap information before the deployment of the device.

(2) Bootstrap from Smartcard In the case that the device requires a smartcard for bootstrapping, a secure channel should be established between the SmartCard and the LwM2M device. When the retrieval of the Bootstrap data is successful, the bootstrap data from the Smartcard must be processed by the LwM2M client and should apply the bootstrap information to its configuration for the purpose of enhancing security.

(3) Client Initiated Bootstrap In this mode, the LwM2M Client contacts the LwM2M Bootstrap-Server to retrieve bootstrap information. In order to contact LwM2M Bootstrap-Server, the client must be pre-configured with information to connect to the Bootstrap-Server and secure the connection. A perquisite for this step is to have the LwM2M Bootstrap Server Account preloaded with the LwM2M Client and the DTLS/TLS or OSCORE security credentials present to authenticate the LwM2M bootstrapping server as shown in Figure 2.3. This is explained in detail as the Client Initiated Bootstrap is the method that represents the basis of the soltuion in this thesis. The first step is the LwM2M Client sending a *Bootstrap-Request* operation to the LwM2M Bootstrap-Server URI. The request should contain the LwM2M Client's Name as a parameter to allow for proper provisioning of information for the LwM2M Client. The next step is to configure the bootstrap information of the LwM2M Client uisng the *Write* and/or *Delete* operations by the LwM2M

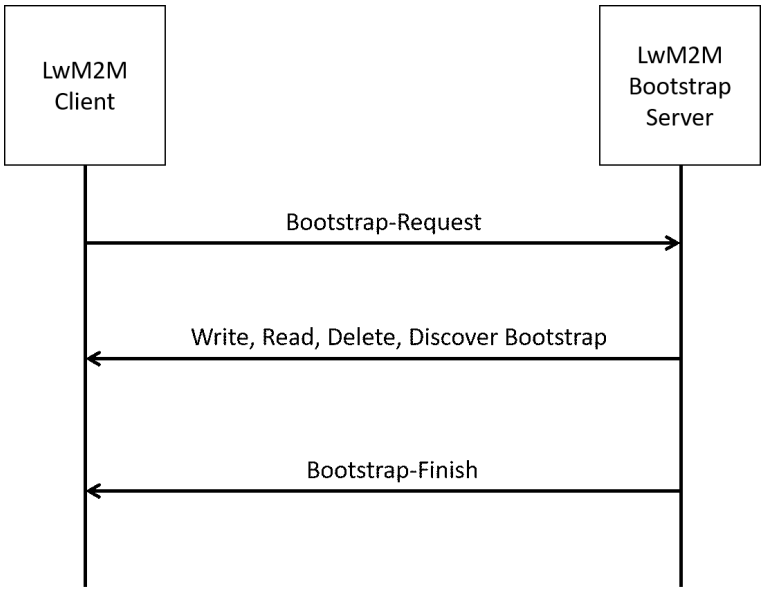

Figure 2.3: Client Initiated Bootstrap [4]

Server. When the LwM2M Server has finished sending the bootstrap information to the LwM2M Client, the server MUST send a *Bootstrap Finish* to the Client to end this phase. Finally, if the process was successful that means that the *Bootstrap Finish* has been received by the LwM2M Client and the configuration is dependable by the LwM2M Client. However, if the Bootstrapping was unsuccessful, the Bootstrap-Server must retain the values that it had before the unsuccessful attempt had occurred.

(4) Server Initiated Bootstrap In the Server Initiated Bootstrap mode, the Bootstrap sequence is initialized by an authorized LwM2M Server. The process is as follows: the LwM2M Server triggers the LwM2M Client to go in the *Client Initiated Bootstrap* instead of another protocol for setting up the Bootstrap information from the LwM2M Client.

When it comes to the *Bootstrap-Request Trigger* as illustrated in Figure 2.4, it is executed from a resource available in the Server Object Instance. This step cannot be performed if there is no connection available between the LwM2M Server and the LwM2M Client in order to enter this *Server Initiated Bootstrap* mode.

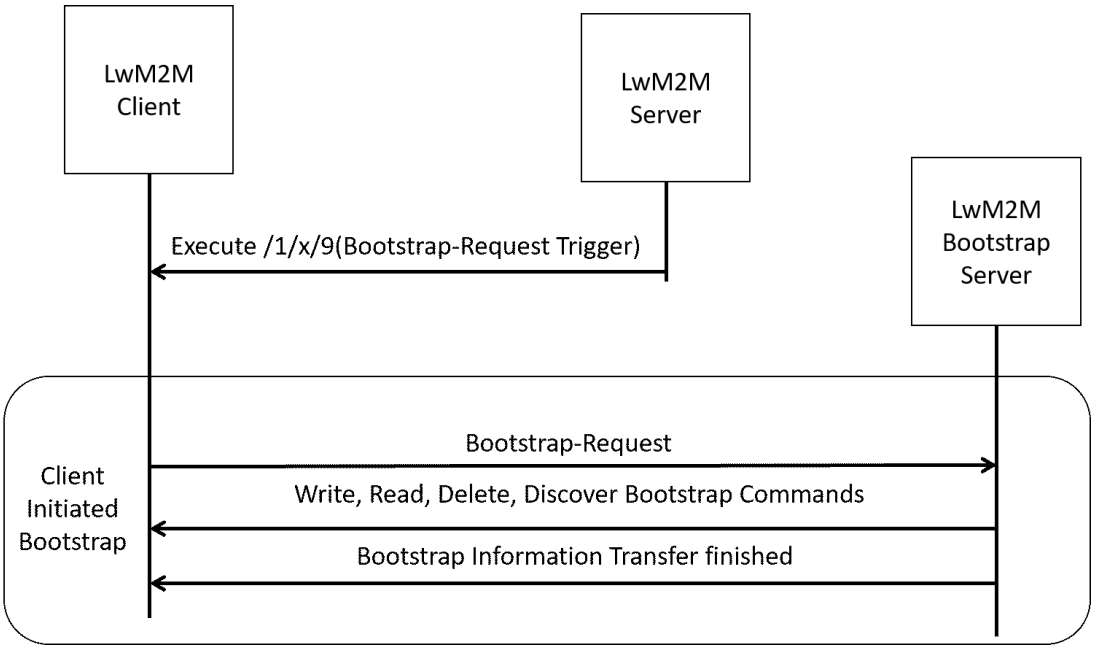

Figure 2.4: Server Initiated Bootstrap [4]

2.5.2 Apple Homekit

Apple Homekit is Apple's solution for consumer IoT network. It is a framework that allows users to control various devices. HomeKit is the app that provides an user interface to coordinate and control home automation accessories. There are various HomeKit compatible devices on the market and they range from but not limited to: lights, switches, thermostats, cameras and doorbells [23].

The pairing process may vary with regards to the different certified devices, however, all of them depend on transferring static codes over OOB channel [12]. The serial code exists on the device itself or on the packaging of which the device comes from. An example for this, is Eve Energy (Smart Plug and Power Meter) [24] that provides features such as voice control, power consumption and built in schedules. The device is plugged into a power socket, it lightens up ready to be paired. The user has two options from the Home App, (1) to scan the QR code and (2) to enter an 8-digit pass-code. Both the QR code and 8-digit pass-code are found

on retail packaging.

Apple requires accessories that interact with apple device to be MFi certified. Apple's Inc. MFi program is an abbreviation for *"Made for iPhone/iPod/iPad"* which is basically a licensing program made for developers wanting to work with the previously mentioned Apple devices. When the certification is acquired for the device, the users are able to access the accessory protocol specification and the communication protocol used to communicate with iOS and macOS devices [25]. MFi allows an accessory to prove that it has been approved by Apple to be used with Apple devices. Apple provides two different mechanisms for MFi Authentication: (1) A dedicated hardware based approach where accessories must emded a dedicated authentication chip known as the MFi Chip on the platform. (2) A software authentication method where the accessory receives one time token that can be used to authenticate itself during device activation in Home kit environment .The verification requires that the software authentication is based on accessory-specific token where the developer needs to create a licensee server to request these tokens. The tokens are then supplied to the accessory and finally the tokens are validated through Apple's and the licensee's servers [26].

HomeKit Accessory Protocol (HAP) is utilized by Apple's Homekit to establish a long-term relationship between the iOS device and a HAP accessory. The protocol includes the following phases: (1) the shared key is derived from an 8-digit pass-code. This is referred to as the Secure Remote Password (SRP). (2) MFi proof which is referred to in the previous paragraph, where a token is verified or an IC chip. (3) exchange of long-term public keys and parameters of the pairing entities [26]. The data can be synchronized between the user's iOS device using iCloud backup and iCloud KeyChain as illustrated in Figure 2.6. Apple's Keychain enables the apps to store passwords, keys, certificates and other types of data.

When it comes to using SRP to secure the pairing process as illustrated in

Figure 2.5, the process of generating the key is as follows [26]: SHA-512 is used as the hash function, in case the SRP reference implementation provided by Stanford is used then the function generates the Session Key, K, from the Pre-master Secret, S, which should be changed from Mask Generation Function 1, `t_mgf1()`, to the specified hash function, SHA-512. The Modulus, N, and Generator, g, are specified by the 3072-bit group RFC 5054 [27].

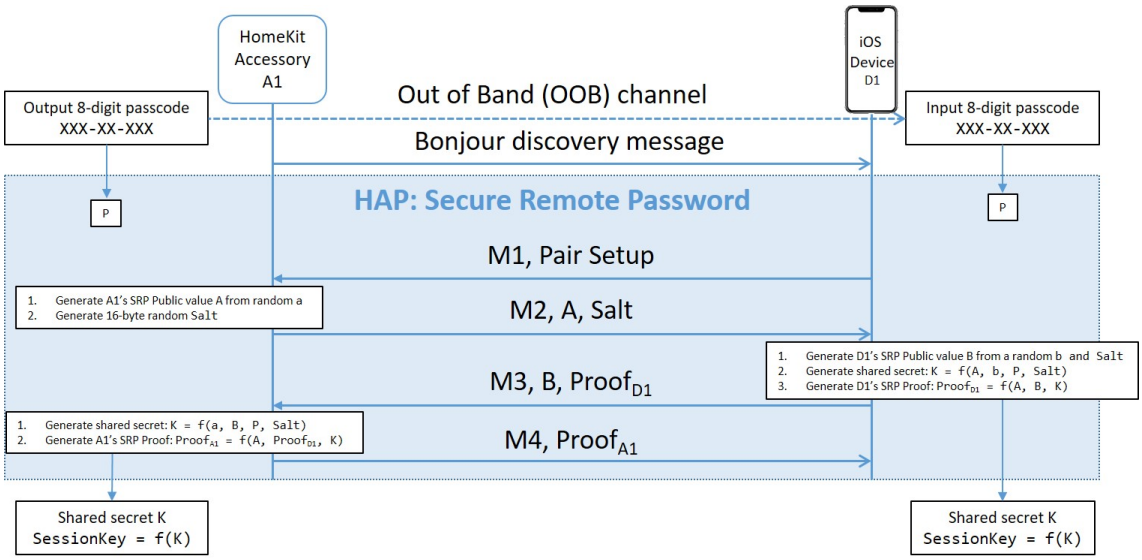

Figure 2.5: Secure Remote Password (SRP) Protocol [28]

In HAP there is a *client* that is always the controller. Its role is visible in sending and receiving responses from HAP accessory servers. In addition to that, the HAP client is responsible for registering and receiving notifications from HAP accessory servers. Another important component in HAP is the *HAP accessory server*, its role is mainly to expose the HomeKit accessories to the HAP controller(s). The HAP accessory server has the following features: it is always the accessory, it accepts requests from clients and sends responses back and it delivers notifications to registered clients. Finally, the *HAP Accessory Objects* which represents the physical accessory on the HAP server. For instance: a thermostat.

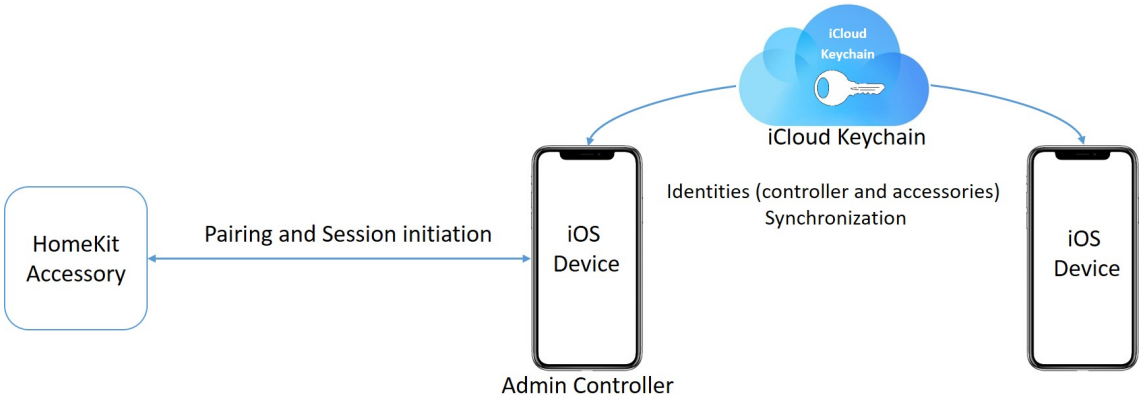

Figure 2.6: HomeKit Data Synchronization between devices [28]

2.5.3 Xiaomi IoT Cloud

Xiaomi is a growing ecosystem due to its relatively competitive price [29]. Xiaomi claims to have the biggest IoT device base worldwide with 85 million devices [30] at an advantage of having different vendors for one ecosystem. The following finding are not published on Xiaomi's official website but in fact were a result of reverse engineering by Giese [31]. Xiaomi's IoT architecture is cloud based. In summary, the Xiaomi cloud works using three main technologies: HTTP, WiFi and Zigbee which is a specification for network protocols [31]. The ecosystem is composed also of a mobile phone containing the smart home app that is used to provision the device with credentials connect it to the WiFi. Eventually when connected, each device has its own connection to the cloud.

Narrowing it more to the connection between the cloud and the device, the following credentials are needed: The *DeviceID* and *Keys*. The *DeviceID* is unique for every device. The *Keys* are the cloud key which is a 16 bit alpha-numeric that is used for cloud communication (AES) and is static. The other key is the token (16 bit alpha-numeric) which is provisioned when the device is connected to a new WiFi Access Point. Moving on to the cloud protocol, the messages are JSON-formatted

messages. A number of updates can be performed over the cloud using this protocol such as APP updates which is the usual software on the device, MCU/WiFi updates and sub-device updates [31].

2.5.4 Google Cloud

Google IoT Cloud [32] allows secure, manageable, and ingested data that is transported from IoT devices to a storage medium i.e Cloud. It supports two main protocols for device connection and communication: MQTT which is an open OASIS and ISO standard lightweight which is common in M2M and HTTP (connection-less protocol). Devices communicate with these protocols using *bridges* such as MQTT bridge or HTTP bridge. The main advantages of both protocols is that MQTT uses lower bandwidth and latency with a high throughput while HTTP is lightweight and causes fewer firewall complexities. One downfall though is that HTTP requires base-64 which exhausts more CPU resources. When it comes to device security, both bridges (MQTT and HTTP) use asymmetric encryption pre-device public/private key encryption with JSON Web Tokens (JWTs). The benefit here is limiting the surface of an attack therefore if the device is compromised, it affects that one device only not the entire network of devices. The JWTs also expires after a certain amount of time so it is limited. The Cloud IoT Core also supports both RSA and Elliptic Curve Algorithms to verify signatures. The device uses TLS1.2 for root certificate authorities, which is required by MQTT. When it comes to *Provisioning Credentials* the *provisioner* which is the user has permissions to create a device. Using the IoT Cloud Core API, the user is able to create a logical device on the cloud. This is illustrated in Figure 2.5.4. The steps are as follows: (1) The key pair is generated by the *provisioner* i.e. user, (2) the *provisioner* initializes the device using the Cloud IoT Core API and specifying the public key which is used for verification, (3) The device manager sends out a positive response that the device was created and

finally (4) The private key is saved in the device for authentication where the Hardware Trusted Platform Module (TPM) is used. As for the Authentication, Figure

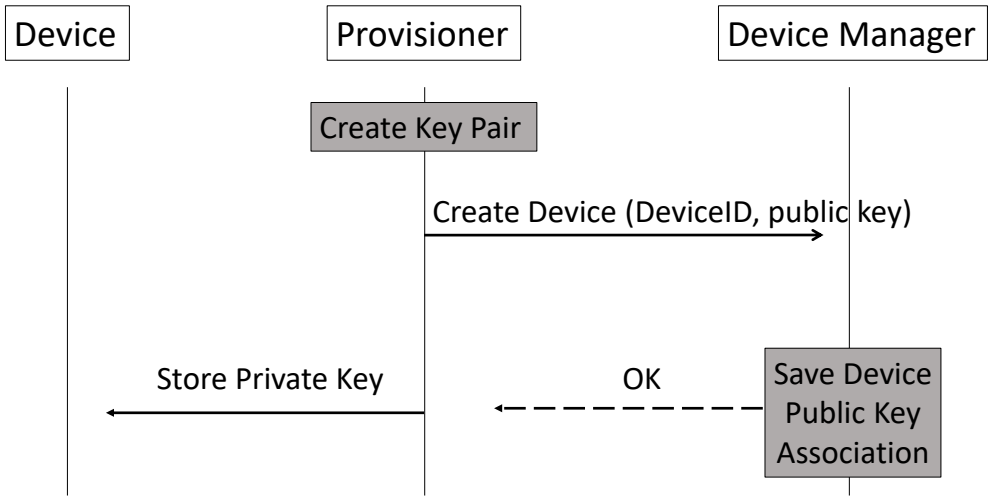

Figure 2.7: Provisioning Credentials [32]

2.8 summarizes the process in Cloud IoT Core with the help of MQTT protocol. Firstly, (1) The device creates a JWT which is signed with a private key (2) Upon connecting to the MQTT bridge, the device shows the JWT as the password in the *Connect* message. (3) The MQTT performs the verification on the JWT using the public key of the device. (4) MQTT bridge accepts the connection and finally (5) When the JWT expires, the connection is immediately closed.

2.6 Android Secure Storage

The security of android devices has grown to be critically important due to vulnerabilities that were found in the Android design. In this section, the hardware-backed secure storage of an Android System is further looked at. If the device contains em-

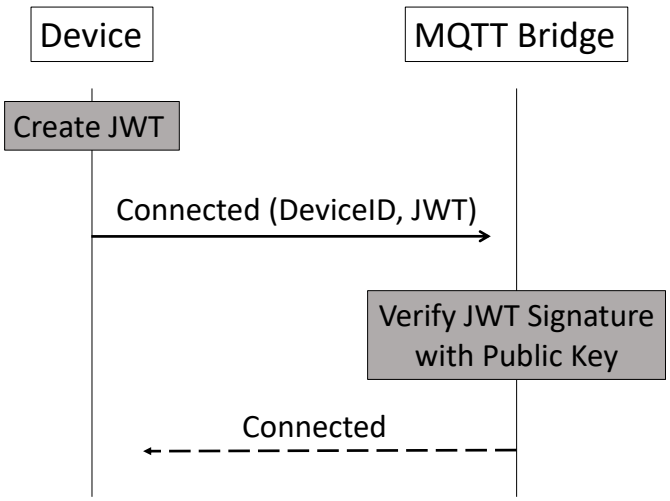

Figure 2.8: Authentication [32]

bedded secure hardware such as a TEE, the keys are safely stored there and provide extra protection.

2.6.1 Android KeyStore

The Android KeyStore was first introduced in API 14 [33] and it communicates with the service *keystore* which uses Inter Process Communication (IPC). The *keystore* is launched upon device boot. It allows for storage of cryptographic keys in a place where it is difficult to extract from the device. As the keys are in the *keystore*, it allows for use of cryptographic operations with the key materials being non-exportable.

The keys are protected by never entering the application processes; the plaintext, ciphertext and messages to be signed enter the system processes which carries out the necessary cryptographic operations. In case an attacker is able to gain access

to the app's process, the app's keys might be used but the keys material cannot be viewed or extracted. This is because the key material is bound to the secure hardware for instance TEE, Secure Element (SE) of the device. This ensures that the key material is not exposed outside the hardware.

There are two variants in the Android Key Store, the first being *Software-based* and *Hardware-based*. Where it means that either software or hardware security features are utilized to protect the keystore respectively. In the *Software-based* variant, some ARM chips do not contain the TrustZone support, while in other cases, the manufacturers are not able to create applications to support TEE. A software equivalent to *KeyMaster* trustlet is created.

A *provider* can be defined to perform cryptographic operations. A popular example of this is *Bouncy Castle*, a collection of APIs used to provide cryptographic capabilities to the Java environment. There is a repackaging for Bouncy Castle called Spongy Castle used for Android. However, the support and maintenance on Spongy Castle has been stopped, and when attempted to apply it to the implementation of the HomeApp, Bouncy Castle functioned just as fine with Android.

2.6.1.1 Trusted Execution Environment

A Trusted Execution Environment (TEE) enables secure execution of application. It is based on a Trusted Platform Module (TPM) and a trusted kernel. Applications running inside the TEE are called *Trustlets*, they are designated to run specifically inside a *trustbox*. The trustlets define an interface that specifies which data can transfer between the trustbox and the trusted world [34].

According to Vasudevan, Owusu *et al.* a number of requirements are provided to ensure a proper Trusted Execution Environment, a more technology-neutral approach [35]:

- Isolated execution: application should be allowed to run in isolation from

other applications in TEE. This provides secrecy and integrity of the code and sensitive information during run-time.

- Secure storage: this is provided using file system permissions, naturally enforced by the operating system. Secure storage offers the ability to maintain the secrecy and integrity of binaries that represent the application when not in run-time. Upon compromising the OS or removing the storage media, such files can then be accessed directly.

- Remote attestation: is allowing verification of the authenticity of a specific message from a particular software module that implements TEE.

- Secure provisioning: allowing data to be sent to a specific module or software on a specific device while providing secrecy and integrity of the data being communicated.

- Trusted path: in order to allow for the communication of the execution environment with the outside world while maintaining the authenticity of data, one hand a party must verify the source of the communicated data from the execution environment. Trusted path can allow for secrecy and availability.

2.6.2 Permissions

According to Android Developer official website[4], before Marshmallow (API 23) permissions were much simpler and were all handled during install-time. Upon downloading an application from the Google Play store the user is shown a list of permissions that are required for the application in which he either accepts all permissions or does not install the app anymore. It was not possible for the user to grant/revoke specific permissions after the application was installed.

[4]Android Developers: developer.android.com

1. Install-Time Permissions: also known as *Normal Permissions* are certain permissions that are categorized to be safe and possibly can't do much harm. These permissions are automatically granted at install time and are never prompted to the user during install time. Install-Time Permissions include: `ACCESS_NETWORK_STATE` and `INTERNET`.

2. Run-Time Permissions: other permissions that are not listed under Install-Time permissions. The user needs to deal with Run-Time permissions by allowing or declining that permission(s) that is requested. The Run-Time permissions usually show up as a dialogue. An example for this is: `READ_CONTACTS` permission.

A paper written by Fang, Han, and Li [36] touched into the topic of permission based android security and the possible vulnerabilities. Coarse granularity of permissions is where an application is provided with arbitrary access to certain resources. Taking into example an `INTERNET` permission, this permission allows full control of the internet access of the application including the ability to send HTTP requests to all domains and connect freely to any destination. As much as the `INTERNET` permission is necessary for many apps, the users are unable to control or restrict its usage. A malicious application could camouflage and legitimize its purpose in order to abuse the `INTERNET` permission. One aspect to view the counter measure for this issue is to create a much more flexible and fine-grained permission models, these could be context-aware models. It could be useful to deploy standard access control policy languages for example XACML to translate Android Permissions. The benefits of this outweigh the degradation that might occur.

3 Architectural Design and System Components

Chapter 3 describes the architecture and design of the system. Section 2.1 of this thesis presented the life cycle model of IoT devices. Figure 3.2 is an adaptation of the life cycle of the IoT device from the above definition. The highlighted aspect of the life cycle of a simple device in this thesis is the bootstrapping procedure and the operational mode of the simple device.

The different components of building the system are discussed along with their relationship with each other and the information exchanged at the different stages of the life cycle. The discussion includes details of the two components of the system that are the LwM2M Client and LwM2M Server. In addition to the different interfaces and operations performed and finally the Android infrastructure and security aspects of it.

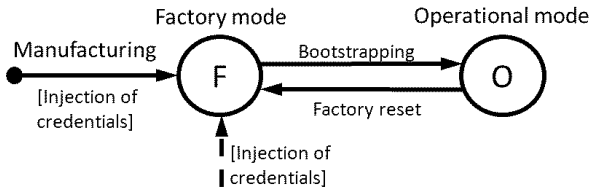

Figure 3.1: Life cycle modes of a simple device [28]

3.1 System Architecture

In this section, a brief description of the proposed system architecture including the simple device, Controller and their structure is explained[1].

The overall system architecture of the proposed solution is presented in Figure 3.2 including various entities and the interfaces available. The Controller entity can include one or more of the following three entities:

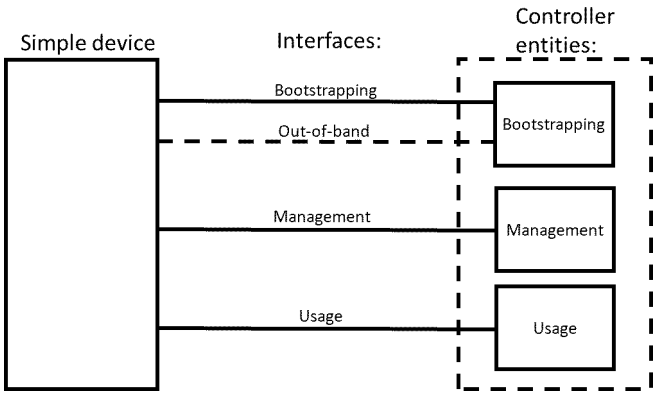

Figure 3.2: Architecture [28]

- Bootstrapping entity: in this phase, the bootstrapping occurs between the simple device and the controller in factory mode.
- Management: during operational mode the management interface is used to control the simple device.
- Usage: the usage interface is used to access the service offered by the simple device.

It is important to point out that the entities (bootstrapping, management and usage) do not necessarily have to exist in the same physical device. The controller entities

[1]The proposed architecture is based on Huawei-internal document [28]

can be distributed across different locations. For instance, the management can be on a remote server, while the bootstrapping can be on a mobile phone.

3.1.1 Simple device

The simple device represents the entity that controls its features and provides access to these features to an external party or a control entity. The main function of a simple device is to authenticate the external party and create an access control point to access the features. In this framework the Simple Device acts as an LwM2M Client.

According to Figure 3.1 which is an applied diagram of Figure 2.1 in Section 2.1. The initial step is the *Manufacturing* of the simple device where credentials, specific device keys and root anchors are injected during the manufacturing process. Following is *Bootstrapping* to ensure the Controller Bootstraps with the correct device which is intended by the user. Finally in the *Operational Mode* the simple device is managed by the Controller with a secure communication based on operational credentials.

3.1.2 Controller

In this framework the Controller runs the LwM2M bootstrap server and the LwM2M Server. The Controller is responsible for bootstrapping and managing the device. Further details are discussed in the upcoming sections about its functionality.

3.1.2.1 Bootstrap Entity

Bootstrapping entity communicates with the simple device over the Bootstrap interface. It is mainly responsible for the bootstrapping procedure when the simple device is in factory mode. After the bootstrapping is successful, the device switches to operational mode.

The role of the bootstrap entity is to use the device credentials of the simple device to enhance the security of the communication between the bootstrapping entity and the simple device. The simple device credentials may include but are not limited to: Symmetric key, Asymmetric key pair, Asymmetric key pair and device certificate. When using the device certificate as a credential, the authentication is provided via OOB channel. The simple device certificate provides additional means to authenticate the simple device remotely.

Other credentials are the operational credentials that are provisioned during the bootstrapping procedure. These include but are not limited to: Asymmetric key pair, Certificate, Symmetric key, key identifier (created during the bootstrapping and must be unique) and Trust roots. Operational credentials are used to perform authentication of device and entities and negotiate authenticated key agreement to provide confidentiality and integrity protocol in communication.

The certificates issues to the devices and trust roots together form an end user specific PKI. Where the user's devices are able to authenticate each other.

3.1.2.2 Management entity

Management entity communicates with the simple device over the management interface. During operational mode, the management entity manages the simple device.

3.1.2.3 Usage Entity

The usage entity communicates with the simple device over the usage interface. During operational mode, it is responsible for using the simple device.

3.1.3 Interfaces

There are four main interfaces in the architecture of the LwM2M enabler illustrated in Figure 4.2 are: (1) OOB, (2) Bootstrap, (3) Device Management and Service Enablement and (4) Usage Interface. An enabler represents a layer of communication protocol between the LWM2M Server and LwM2M Client in addition to the LwM2M Bootstrap Server and LwM2M Client.

3.1.3.1 Out-of-Band (OOB)

Out-of-Band (OOB) is used to provide additional authentication mechanisms that aid in increasing the security of the bootstrapping interface. In addition to that, it assists in further enhancement of the discovery mechanism between the simple device and the bootstrapping entity.

3.1.3.2 Bootstrap interface

A *Bootstrap Interface* mentioned in Section 2.2 provisions operational credentials such as key pairs, certificate and trust roots. In addition to connectivity settings (Home network details). The LwM2M Client must have the LwM2M Bootstrap information after the Bootstrap sequence. The client must have at least one LwM2M Server account after the Bootstrap sequence and may also contain additional object instances and security object instances [4]. This is used to register the LwM2M Client to the LwM2M Server.

The highlighted steps in Bootstrapping that are emphasized throughout this thesis are:

1. Data transfer using OOB Channel: the communication over the OOB interface involves the user. For example, one of the possible methods to make the OOB transfer of information from simple device to Controller is through QR

Code scanning; when the Controller device scans the QR code from the simple device.

2. Discovery of simple device: the simple device acts as a WiFi access point when it is first powered on. The Controller receives the WiFi network details over the bootstrap interface as stated previously in data transfer in OOB stage point. In operational mode, after successful bootstrapping, the simple device stops acting as an access point and joins instead the WiFi network that the Controller Configured during the Bootstrapping procedure.

3. Establishing a secure connection: this is done using DTLS. After the device discovery, the bootstrapping procedure continues to the DTLS session setup. The DTLS session is initiated by the simple device.

4. Authenticating the simple device and the Controller: this is to make sure that the simple device is connected to the "right" Controller and vice versa. This is done through recieving the simple device credentials when the Controller verifies the device's credentials with the information received over the OOB interface.

5. Configuring simple device: here the device moves from factory to operational mode.

3.1.3.3 Management Interface

The Device Management and Service Enablement Interface are used by the LwM2M Server to gain access to the Object Instances and Resources present in the LwM2M Client. The Operations available for this interface are: *Create*, *Execute*, *Delete*, *Write*, *Write-attribute*, *Read*. *Read-attribute*. *Delete* and *Discover*.

3.1.3.4 Usage interface

Usage interface is used to operate the simple device. It is responsible for operations related to the features of the simple device.

3.2 Leveraging LwM2M features

Here the LwM2M features are leveraged for the simple device during the life cycle management process.

3.2.1 LwM2M Client

The LwM2M Client in Figure 3.3 assists as an end point of the LwM2M protocol that communicates with the LwM2M Server in order to execute operations from the LwM2M Server to the device such as Device Management and Information Reporting.

3.2.2 LwM2M Bootstrap Server

The LwM2M Bootstrap Server provides the LwM2M Server account information by updating the security object instance.

3.2.3 LwM2M Server

A LwM2M Server is a logical entity that sits within the Machine to Machine Service Provider and represents one of the end points of the LwM2M communication protocol.

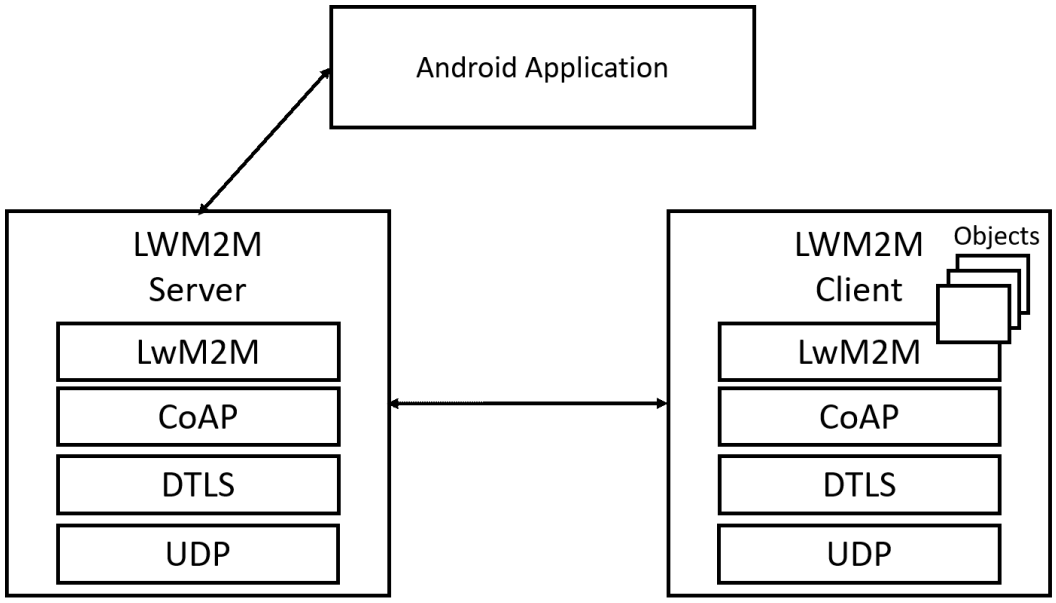

Figure 3.3: Overview of the Target System

3.2.4 Realization of Architectural interfaces

Here the architectural interfaces are realized using LwM2M interfaces. This is also a more specified description regarding the choices made in this thesis to adapt to the standards of LwM2M.

3.2.4.1 OOB

Communication over OOB typically involves the user, to different degrees according to varying scenarios. The involvement could be the user scanning a QR code on the simple device with his/her smart phone's camera which in turn binds the simple device to the Controller.

The user turns on the simple device, starts the QR reading functionality on the Controller device and attempts to scan the QR code from the simple device which encodes the device's unique identifier, Wifi SSID and other data via the OOB channel. The controller then joins the WiFi network of the Access point (AP) in the simple device.

3.2.4.2 Bootstrap interface

The bootstrapping procedure involved configuring the simple device by the Controller and moving the simple device from a factory reset state to operational mode. This is the standard bootstrapping procedure based on LwM2M specifications [4]. Two main additions are introduced in this thesis: during bootstrapping, (1) a process of transferring a unique identifier of a simple device through the OOB channel to the Controller. This unique identifier could be: PSK, PK or certificate. (2) Binding of the simple device's credentials that are received by the Controller over OOB, to be able to communicate through an in-band Channel such as WiFi.

After device discovery, the bootstrapping process is resumed to a DTLS session setup which is initiated by the LwM2M Client (simple device). More is discussed on client initiated bootstrap in Section 2.5.1.1. After a successful DTLS handshake, a confidential and integrity-protected channel is achieved over the in-band interface.

The bootstrapping steps along with the additions can be summarized to the following points:

1. Discovery of the simple device by the Controller.

2. Creation of Credentials and integrity protected Channel between the Simple device and the Controller.

3. Channel binding which means that the Controller paired with the right simple device through correct credentials.

4. Agreement on long-term keys and other operational credentials

3.2.4.3 Management interface

The management interface may include the possibility to add, modify and remove operational credentials and trust roots. It may also include initiating a factory reset

of the simple device. A client registration interface can be used. The Client Registration interface shown in Figure 3.2. The LwM2M Client must be able to support the following operations: *Register*, *Update* and *De-register* operations. While the LwM2M Server must support all operations on this interface. The Client Registration interface is used by an LwM2M Server to register with one or more LwM2M Servers. The Register-Operation is used by the Client while providing the necessary information for the LwM2M Server while maintaining the registration and communication sessions with the LwM2M Server on the basis of configuration parameters, for instance the Que Mode or Lifetime. Finally, the LwM2M performs a *De-register* operation upon discontinuing the use of a LwM2M Server.

3.2.4.4 Usage interface

In the usage interface, operations that are related to the features of the device are carried out. An interface that can be used as a usage interface is the Information Reporting Interface. It observes alterations in a Resource(s) belonging to the LwM2M Client. Upon addition of new values, notifications are received by the LwM2M Server. This is achieved by sending either one of the operations *Observe* and *Observe-Composite* to the LwM2M Client to gain information about an Object, Object instance or a Resource. To terminate this operation, the LwM2M Server sends a *Cancel Observation* or *Cancel Observation-Composite* to be performed. It is important to note that the LwM2M must ignore any Observation Operation if it has not received a Registration acknowledgement.

4 Implementation

Chapter 4 includes the implementation on the basis of the system architecture in Chapter 3. Figure 4.1 summarizes the solution proposed in this thesis for securing the bootstrapping and registration of the simple device using a Controller.

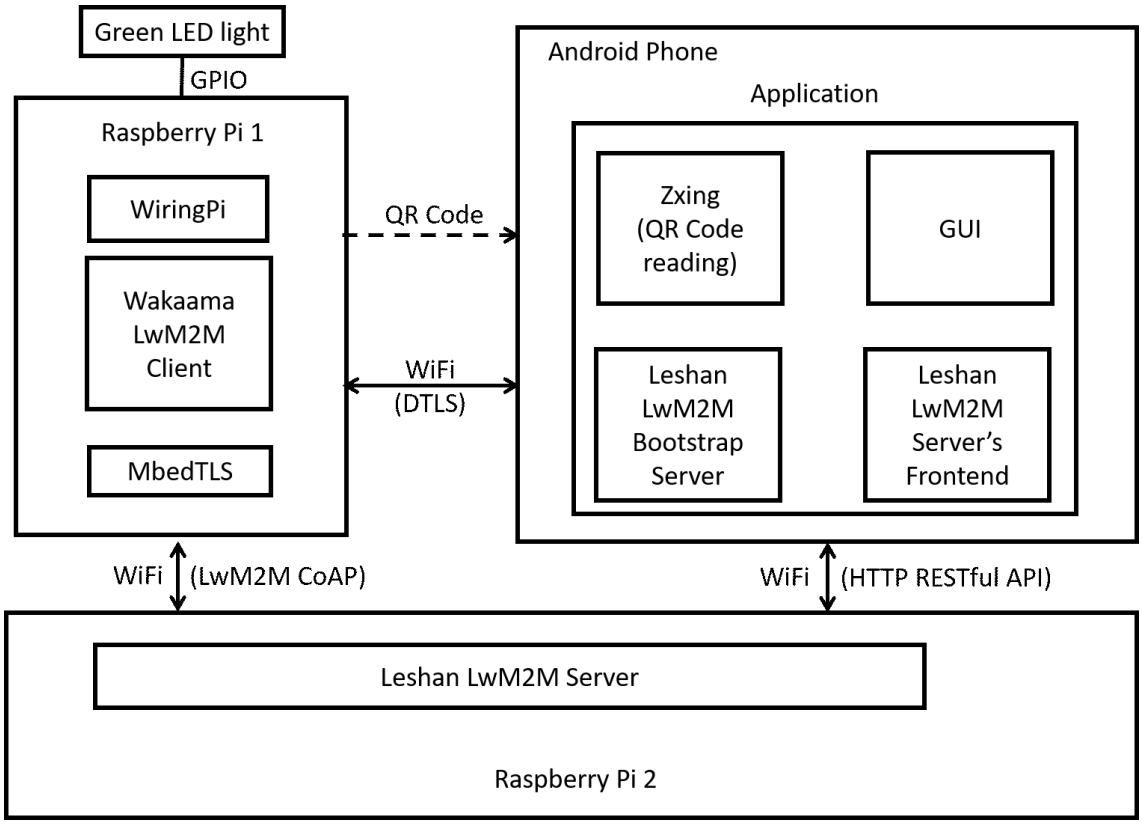

Figure 4.1: Experimental Setup

4.1 Simple device

The LwM2M Client is implemented on a Raspberry Pi 3 Model B+ using MbedTLS open source library for DLS/TLS in C, more details are found in the following master's thesis [37]. The main functionality of Raspberry Pi is acting as an access point to the Controller to provision bootstrapping credentials to the simple device. More are details discussed about provisioning of credentials is found in Section 4.5.

4.2 Controller device

Leshan is an OMA LwM2M server and client Java implementation library [38]. In this setup, Leshan is used to implement the LwM2M Server and the LwM2M Bootstrap server.

The LwM2M enabler is discussed in detail in Section 3.1.3. It defines the application layer communication protocol between the server and client. The following logical interfaces are defined for this thesis:

1. Bootstrapping
2. Device Discovery and Registration
3. Device Management and Service Enablement

Information reporting is not yet implemented in this phase of the thesis but is a part of future development.

For setting up the environment, Android Studio is used with Java. The compiled SDK version is API 28 intended to run on *Huawei P30 pro* test phone. The graphical user interface (GUI) was sketched using *MockFlow* [39] which was later implemented using Android studio layout features as xml files.

There are three main configurations to run the framework:

1. Running the Server and the Bootstrap Server in the Android application alongside the LwM2M Server and Bootstrap Server as a whole system in the same device.

2. Dedicating a gateway (home router) where the Android application runs the Server and the Bootstrap Server while the LwM2M Server and LwM2M Bootstrap server runs on a home router in the same network.

3. Android app runs the Server and the Bootstrap Server while the LwM2M Server and LwM2M Bootstrap server run on a cloud service which the user trusts.

For this thesis, option (2) is used. Option (1) is a part of future enhancements.

4.2.1 Bootstrap Server and LwM2M Server

The Bootstrap Interface consists of a set of operations used to initialize the required objects to the LwM2M Client to register, such as uplink operation which includes *Bootstrap-Request* and downlink operations under the names of *Discover*, *Write*, *Read(restricted)*, *Delete* and *Bootstrap-Finish* illustrated in Figure 4.2.

The Bootstrap Operations used in the Bootstrap Server in further details are as follows: (earlier description found in Section 3.1.3.2:

1. Bootstrap-Request Operation: used to initiate a Bootstrap sequence during *Client Initiated Bootstrap* which is a Bootstrap mode that retrieves Bootstrap Information from the LwM2M Bootstrap-Server Account.

2. Bootstrap-Finish Operation: in order to terminate a Bootstrap Sequence that was previously initiated during the *Client Initiated Bootstrap* mode. The main goal of this operation is to notify the LwM2M Client that the necessary Bootstrap information have been provided by the LwM2M Bootstrap-Server.

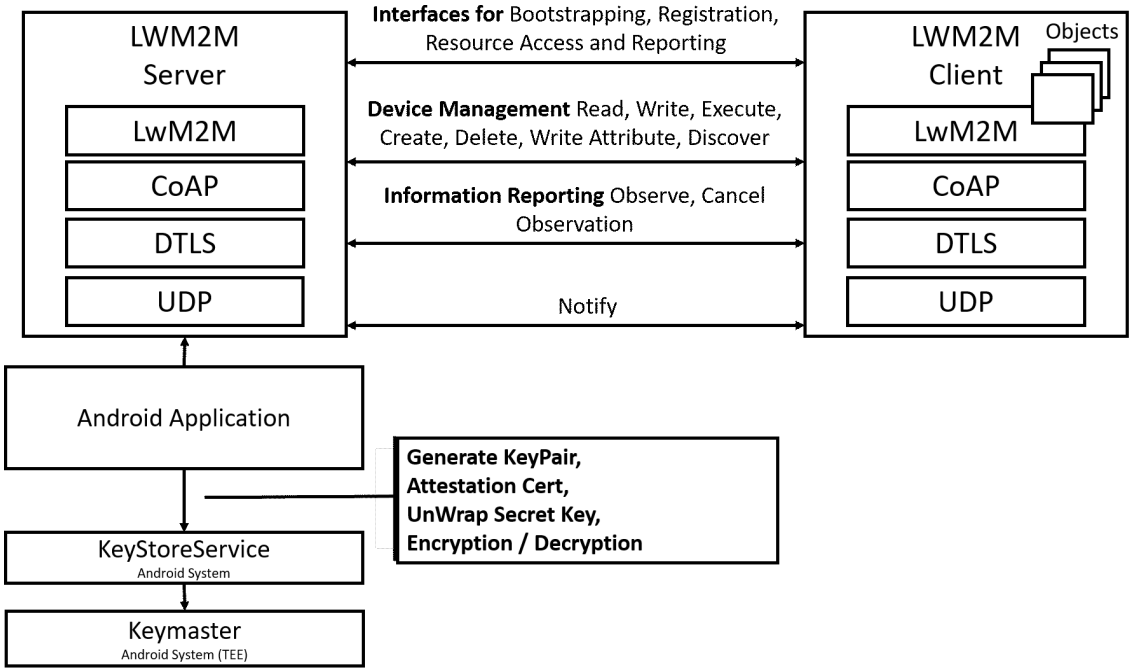

Figure 4.2: Interaction between Android, LwM2M Server and LwM2M CLient

3. Bootstrap-Discover Operation: the main purpose of this operation is to recognize LwM2M Objects and Object instances that are supported by a specific LwM2M Client. It is not required to specify the Object ID, a targeted path is accepted instead of an Object ID. This notifies the Client to show all existing Object instances.

4. Bootstrap-Read Operation: The read operation must be restricted to include the Objects required to perform a proper configuration of an LwM2M Client. These objects are: LwM2M Server Object (Object ID: 1) and Access Control Object (Object: 2). These allow the Bootstrap Server to validate the present server accounts and add new ones without violating the present Access Control rights in the LwM2M Client.

5. Bootstrap-Write Operation: this operation can be sent multiple times. In

the Bootstrap-Write operation if the LwM2M does not support the optional resource, it must not instantiate Objects for those but must silently ignore.

6. Bootstrap-Delete Operation: like the Bootstrap-Write operation, this can be sent multiple times. The Bootstrap-Delete can be used with any instance or all instances of any Object which is supported by the LwM2M Client.

To set up the LwM2M Server and the Bootstrap Server, the implementation provided by Leshan is used. In order to integrate it with Android, the Leshan library modules are compiled into jar files that are loaded as libraries to be used for the Server and Bootstrap Server implementations. Once that is done, a Leshan UI sandbox is available for testing using `http:\\Server-IP_addr:8080` to run in *local host* or through testing the LwM2M Server at `https:\\leshan.eclipseprojects.io\` and the Bootstrap Server at `https:\\leshan.eclipseprojects.io\bs\` these web UIs allow easy management of LwM2M client devices. It is important to note that for authentication, X.509 certificate authentication is used for Bootstrapping and Server protocols through proximity authentication. However, the LwM2M Server certificate is ignored by the LwM2M Client.

4.3 OOB

Out of Band (OOB) interface is used to provide an additional authentication method to further secure the bootstrapping interface. Another usage of OOB, is to enhance the discovery mechanism between the simple device and the Controller before the execution of the bootstrapping attempt. OOB channel is exhibited when the QR code sticker is seen on the simple device, then the Controller attempts to bootstrap by capturing the QR code using the mobile device's camera.

Three bootstrapping techniques are used in the Android application on the basis of the discussion in Section 2.2. The bootstrapping method chosen for this prototype

is *Client Initiated Bootstrap* which is explained in detail in Section 2.5.1.1, which shortly allows for the LwM2M Client to retrieve bootstrapping information from the LwM2M Bootstrap Server which is situated in the Controller.

1. **QR Code Scanning:** As explained in Section 2.2 QR code is the most popular method used nowadays to gain control or access different types of information, which is the reason why it is considered the primary OOB channel method to be offered to the user. The QR code is implemented using ZXing Android Embedded Library[1] which is based on ZXing Android Barcode Scanning Application. This library is used because it is easily integrated with Android and provides advanced customization of the logic and UI. Processing wise, the camera launched during QR code scanning is managed in a background thread which reduces the start up time, therefore reducing the waiting time for the user. The version used for this prototype is 4.1.0 which requires SDK 24+.

[1]ZXing Android Embedded Repository: github.com/journeyapps/zxing-android-embedded

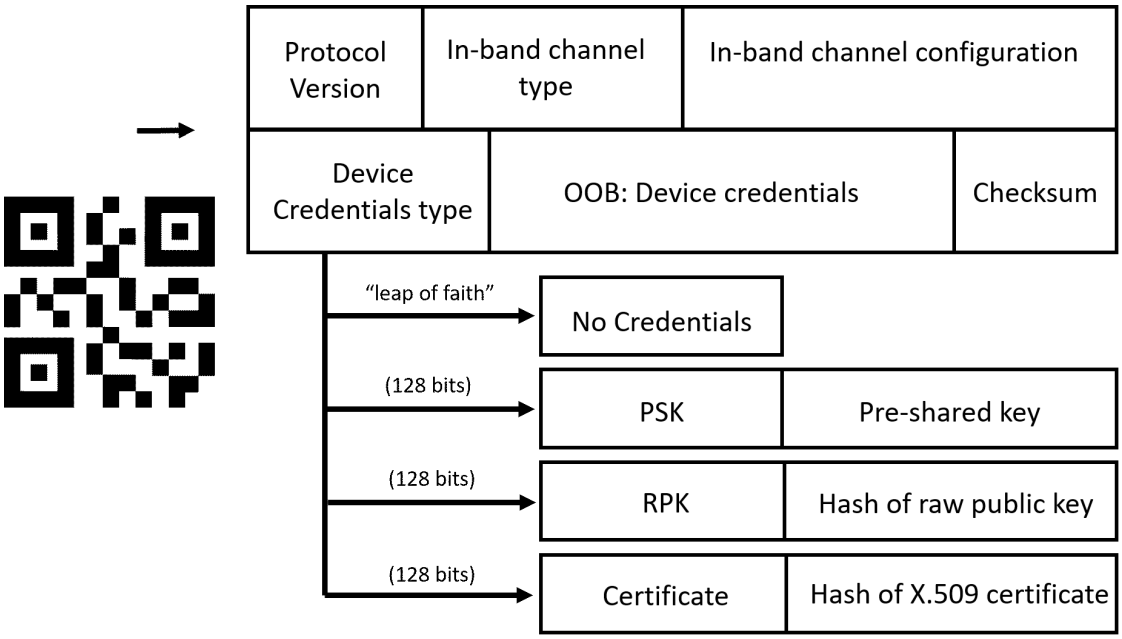

Figure 4.3: OOB Message Structure [37]

The hash encoded in the QR code is 128 bits in length using SHA256 digest of the LwM2M Client Certificate. The QR Code is then printed and placed on the IoT device and its package to be used during the bootstrapping process. The format of the QR code message is illustrated in Figure 4.3. This hash is used for the purpose of illustrating the message structure:

```
01;WIFI;WPA2-PSK:huawei_iot:c1e7d64c383e;03;367f36ee1f5e272cfbca1
ca35a73e42b;34;;.
```

Below are the details of the encoded information:

(a) Protocol Version: here 01 is used.

(b) In-band channel type: WiFi is the choice of the in-band channel communication protocol.

(c) Device Credentials type: 00, 01, 02, 03 allocated for the no security (leap-of-faith), PSK, RPK and X.509 certificate respectively.

(d) `WPA2-PSK:huawei_iot:c1e7d64c383e`: is the credentials for the in-band communication channel. This field refers to encryption_algorithm:SSID:PSK. This field also supports 802.11x based authentication.

(e) `367f36ee1f5e272cfbca1ca35a73e42b`: is the left most 16 bytes of the SHA-256 hash output of LwM2M Client (simple device) device certificate.

(f) 34: is the checksum value for manual entry. This is the left most 1 byte of the SHA-256 hash output of the left most 16 bytes of the LwM2M Client (simple device) device certificate.

2. **Optical Character Recognition:** this feature is implemented with the help of Firebase[2] which relies on Machine Learning with an on-device or cloud-based API to recognize latin letters.

3. **Manual URI entry:** the least preferred pairing method as it is prone to user typing errors. Here, the user is given the option to manually type the URI which is then verified in the same way against the certificate received from the LwM2M Client.

4.4 Bootstrapping

This subsection in the thesis aims to answer RQ3 regarding protecting provisioning credentials. To be able to answer that, it is vital to look into the two bootstrapping phases that enable establishing protection and security during bootstrapping and operational modes.

The bootstrapping phase in this thesis has two phases:

- Secure channel establishment: as mentioned in the previous Section 4.3 a hash of the public key of the LwM2M Client is encoded in the QR code and

[2]firebase.google.com/docs/ml-kit/recognize-text

provided over the OOB channel. This is exchanged during a DTLS channel and is used to authenticate the simple device. For verifying the simple device, the Controller must compare the hash of the public key provided during the Bootstrap attempt with the hash of the certificate of the device recieved during the DTLS handshake.

- Operational credential provisioning: once a DTLS channel is established between the LwM2M Client and the LwM2M Bootstrap Server, the LwM2M client will receive provisioning credentials from the LwM2M Bootstrap Server (Controller). The security object mode is set to Certificate mode.

With reference to 4.4 the simple device accepts the WiFi connection and an IP is given to the Controller. A DTLS connection is established as shown in step 8. The simple device sends a bootstrap request to the Client. The Controller provisions server operational credentials that include: a key pair, an X.509 certificate and trust roots and home network details. Both the Controller and simple device switch to home network. It is vital to point that the simple device will restart in operational mode.

4.5 Usage in operational mode

Figure 4.4 illustrates how the different system components interact with each other. The process beings with the user launching the *HomeApp* and adding his/her home network details to the settings. The user turns on the simple device for the first time and uses his/her phone to scan the QR code on the simple device. As seen in step 6, the Controller extracts the authentication details in the QR code (see Section 4.4 for details) and connects to the device's access point.

After the simple device has restarted in operational mode 4.4, the simple device will then send a registry request and from there the Controller registers the device

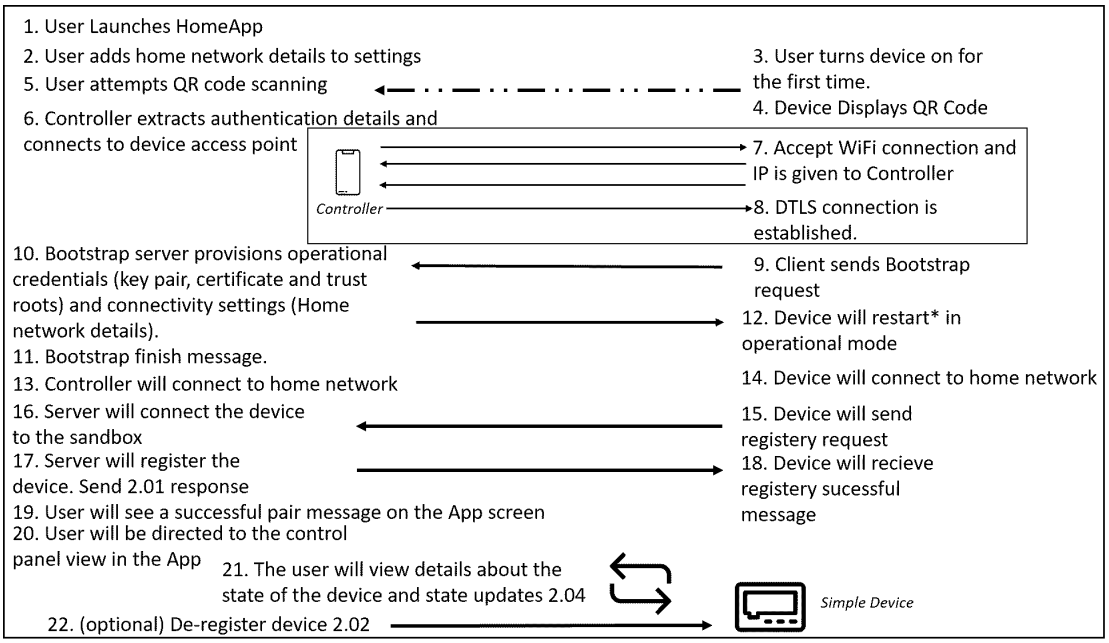

Figure 4.4: System Interactive Diagram

to the server and sends a 2.01 response as seen in step 17. Finally, the user is able to see that the simple device is registered and can view details about the device in addition to controlling its actions.

In order to manage the application with a real life scenario as illustrated in Figure 4.5, an LED light is managed remotely over the network. This is a good scenario as LwM2M is a device management protocol layer which provides different management functionalities. An LED is connected to the RaspBerry Pi device which represents the LwM2M Client. After bootstrapping the device to the *HomeApp* successfully using Leshan's Bootstrap Server, the light source is able to be controlled and dimmed or turned on/off using the control panel in the *HomeApp*. This is done by sending HTTP requests to the web interface provided by Leshan Server which runs on a second Raspberry Pi (could also a use a PC) as shown in the beginning of chapter 4 in Figure 4.1.

In addition to managing the LED light, the status of the light is seen with

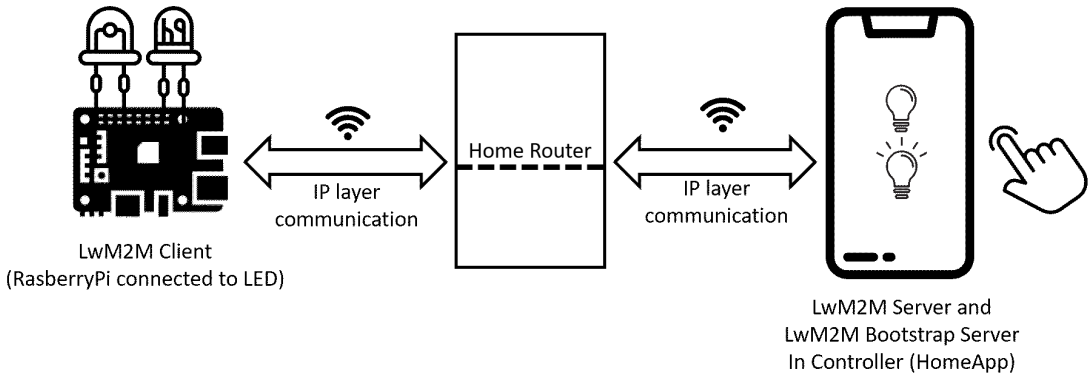

Figure 4.5: LwM2M Application Scenario - Control of an LED with the HomeApp

the help of the *Observe operation* through the LwM2M Server. Upon changing the switch to on or off, the LwM2M server immediately notifies the LwM2M client which received the change of event and the corresponding status was updated on the web UI. An example of such HTTP request is shown in Figure 4.6.

One more important thing to note, in the experimental design setup the Leshan front-end is located in the phone and Leshan Server in the Raspberry Pi 2 communicate over HTTP. Note however, that in the actual home network this communication should be secured, e.g., using HTTP with mutual authentication during a handshake.

4.6 Storing Device Related Data

Shared Preferences in Android allow for storing primitive data or key/value pairs in a file in the mobile device. The main purpose of using shared preferences in the *HomeApp* is to store the Home network details which are sent during Bootstrapping for the device to use to connect to the home network after the Bootstrapping attempt is successful. In order to maintain the confidentiality of these details, the shared preferences must be protected.

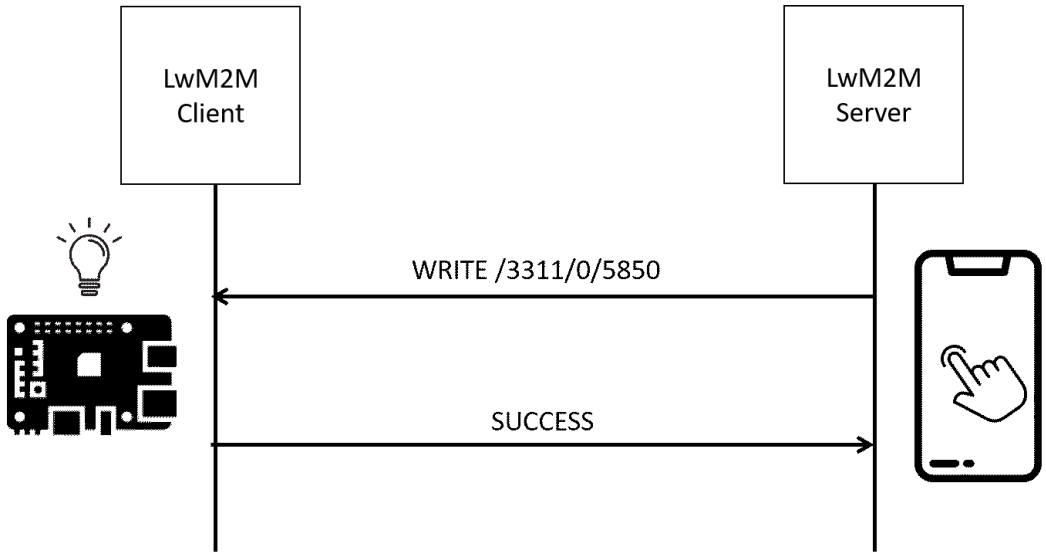

Figure 4.6: Usage Interface

The Android framework provides Encryption of Shared Preferences using AndroidX Security Library, the library runs using SDK 23 and above. An encryption master key is stored in the Android Keystore using `AES256_GCM_SPEC` specification for instance. Then an instance of `EncryptedSharedPreferences` is initialized which is a wrapper around `SharedPreferences`. The Shared Preferences values can then be read and stored securely.

4.6.1 Self-Signed Certificates

Self-Signed Certificates are a type of Certificate that is signed by the same issuer of the certificates whose identity it certified. In other words, self-signed certificates are signed with their own private key.

The issued certificates to the device and the trusted roots together play the role of a Certification Authority (CA) since it uses PKI to implement the access control of the simple device from the Controller entities (See Section 3.1.2). The added difference here to the LwM2M Specifications is that this approach does not require

a database to be maintained where all the trusted peers (Controllers) of the simple device are stored. This is because a peer can be verified from a certificate presented when the peer attempts to connect to a simple device.

A self-signed certificate for the LwM2M Bootstrap Server and Server is generated with OpenSSL, which is a toolkit for TLS and SSL protocols and contains cryptography libraries as well. The following command is run for that purpose `req -x509 -sha256 -nodes -days 365 -keyout privateKey.key -out certificate.crt`.

4.7 User experience

In this section the design of the application GUI is illustrated and its functionality is explained. The drive of this design is to provide a smooth user experience during bootstrapping and registering phases. The application *HomeApp* uses *eight* screens to ease the user experience when connecting to the simple device. The Figures 4.7 and 4.8 illustrate the flow between the screens of the *HomeApp* application. The user starts by viewing the Control Panel upon application launch. From there, the user is able to add accessories, navigate through the list of added accessories and further more. The screens are explained in detail in the following subsections. It is important to mention that the term *Accessory* is used interchangeably with the term *Device* where they both mean the Simple Device. However, for simplicity purposes IoT Device is not used as it may be complicated for the user to comprehend.

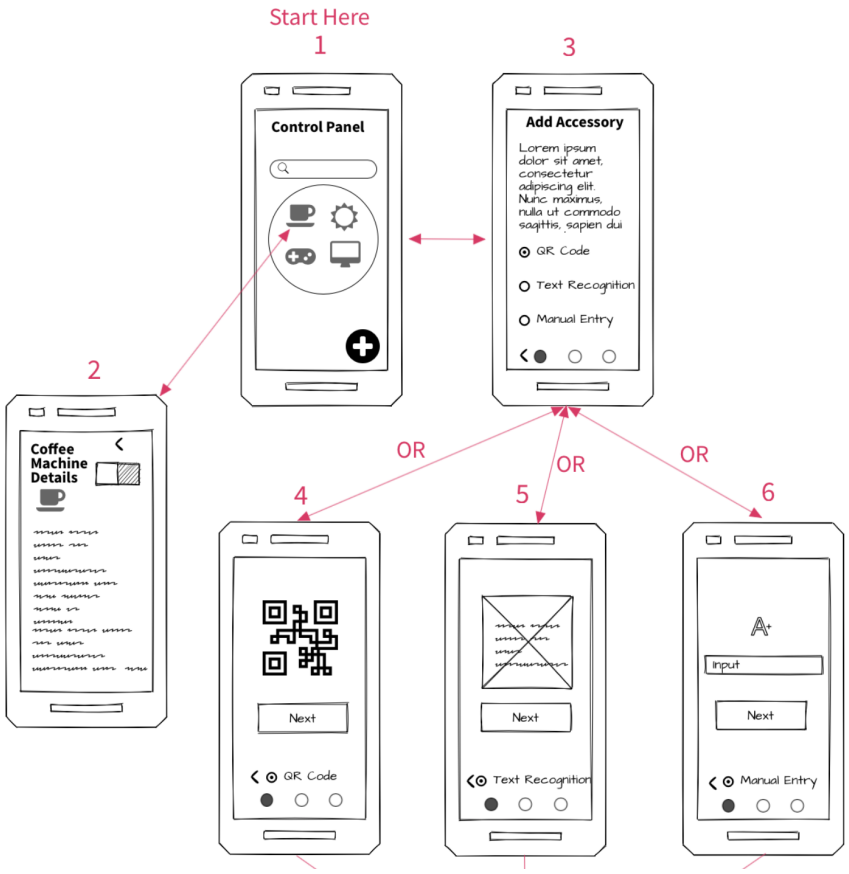

Figure 4.7: Schematic UI Functionality Illustration (1)

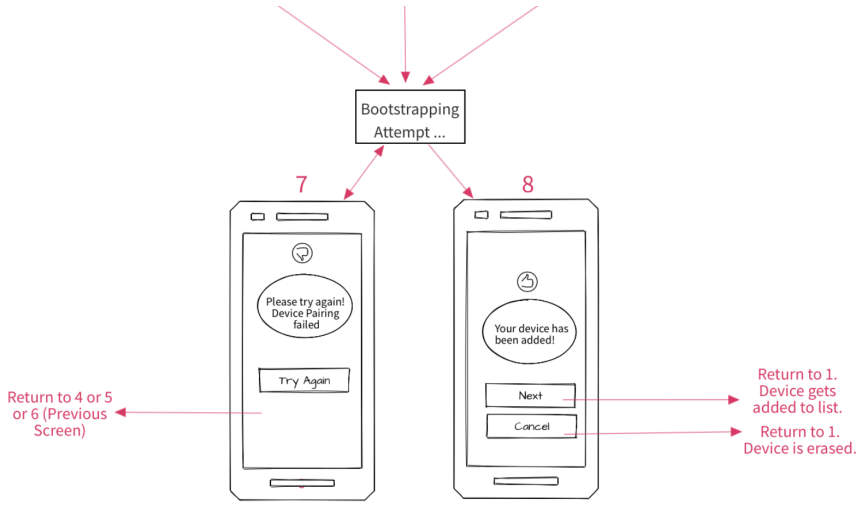

Figure 4.8: Schematic UI Functionality Illustration (2)

4.7.1 Screen 1: Control Panel

The *Control Panel* is visible upon launching the *HomeApp*. The *Control Panel* contains the list of paired accessories connected to the application by the user. The purpose of this screen is to present a panel to ease the control and operation of the accessories from a single view. The user can also view the statistics and important information regarding the paired accessories. An example for this is the battery percentage of a simple device. The screen allows the user to add new accessory as well.

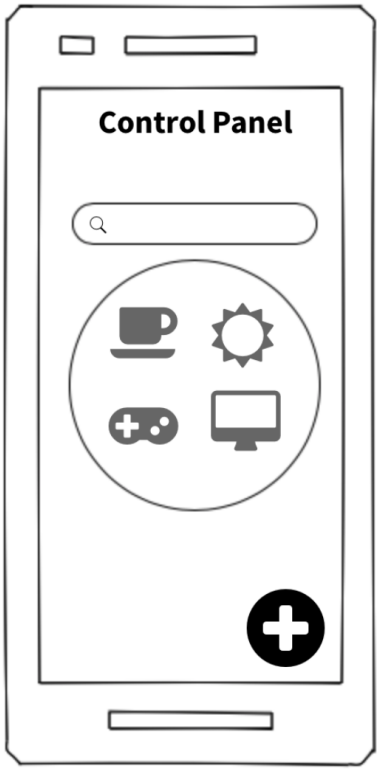

Figure 4.9: Control Panel

4.7.2 Screen 2: Operation and Control of the accessory

This screen is launched upon selecting a widget from the *Control Panel* screen in Figure 4.9. The user is able to view all the information needed about the accessory. In addition to that, the user is able to control the accessory. For instance, he/she can toggle the on and off button.

Figure 4.10: Information about IoT Device

4.7.3 Screen 3: Adding an Accessory

In the context of *HomeApp*, the addition of the accessory corresponds to bootstrapping the accessory and registering it. This *Add Accessory Screen* is launched when the user selects the *plus sign* button from the previous page. Here, the user finds instructions on how to pair the simple device with the phone (Controller).

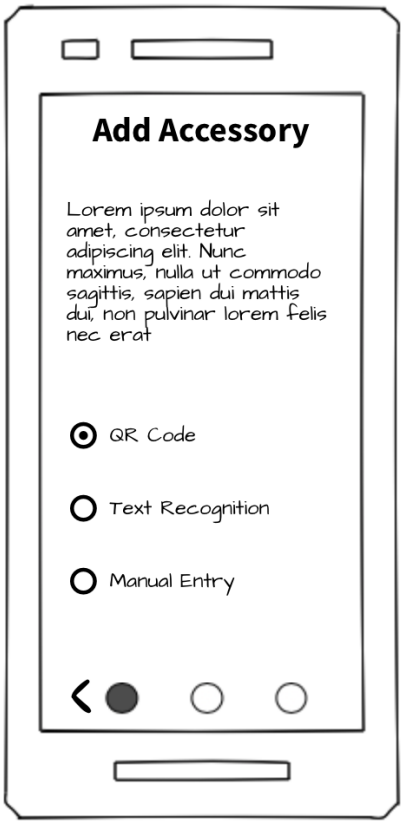

Figure 4.11: Adding Accessory

The instructions include but are not limited to: connecting the device to the power supply in case it is disconnected, turning on the device .. etc. Below the instructions, the user can find three options to pair his/her accessory. (1) Using QR Code, (2) Using Text Recognition to read the URL and finally (3) Manual entry of the URL into the application. *HomeApp* allows the user to swipe back and forth between the pairing methods to choose whichever method is most convenient.

However, the QR code is selected by default as it is the easiest option for the user.

4.7.4 Screen 4: Using QR Code

Here, the user aims the camera at the QR code of the device. It is essential that the QR code is present completely in the frame of the camera screen in order to successfully recognize it.

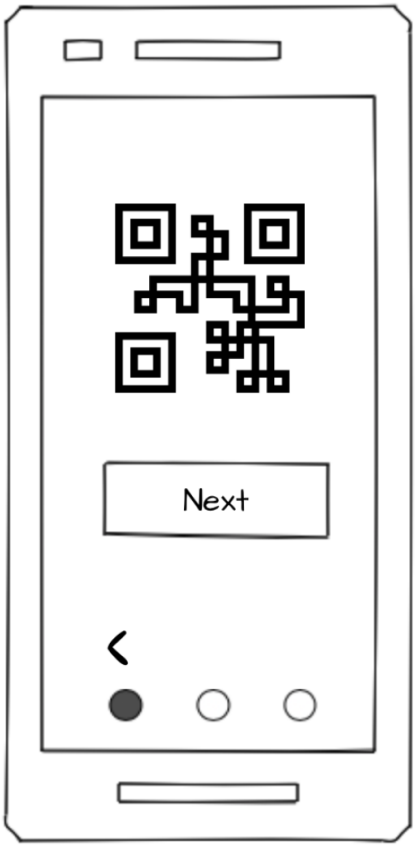

Figure 4.12: QR Code Scanner

4.7.5 Screen 5: Text Recognition

If the user chooses this option, the camera must be aimed at the URL instead of the QR code. The user should make sure that the whole text is captured in the camera frame to ensure a successful process in reading the characters.

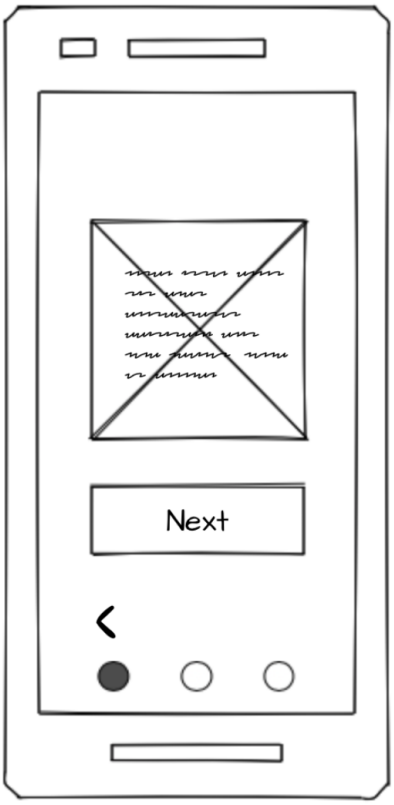

Figure 4.13: Text Recognition

4.7.6 Screen 6: Manual Character String Entry

The user is given the option to manually enter the URL into the application. This option is not recommended as it is prone to human typing errors. However, it may be used in the case when both QR code reading and character recognition fails.

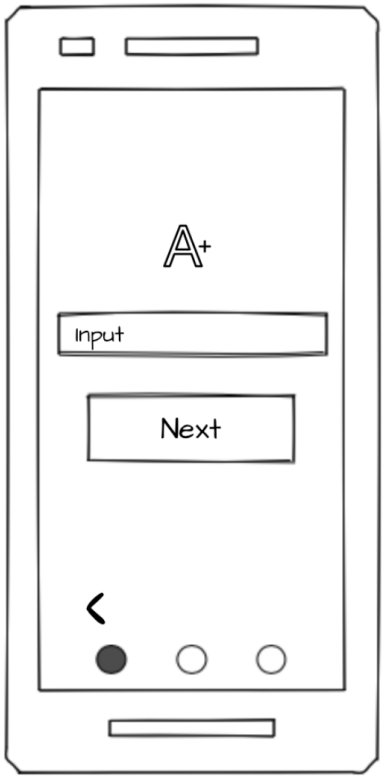

Figure 4.14: Manual Character Entry

4.7.7 Screen 7: Failure to Recognize and Add an Accessory

If the bootstrapping attempt fails, the user has the option to re-attempt through the *Try Again* Button. From here, the user is redirected to the previous screen. See figures 4.12, 4.13 or 4.14. This emphasizes on RQ4, to provide a user friendly experience to the user during bootstrapping by providing a range of options to perform this process.

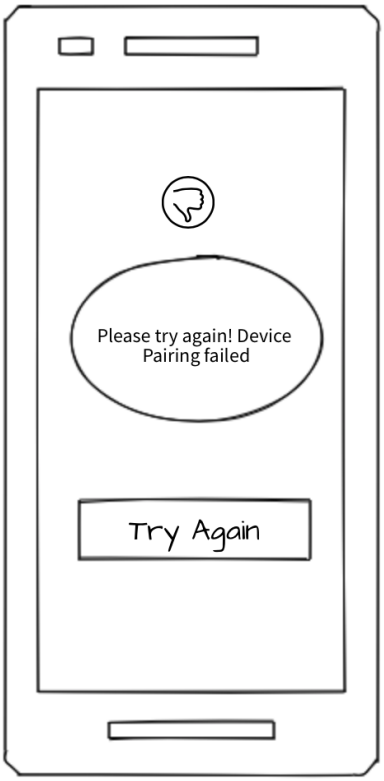

Figure 4.15: Failed Attempt to Add Accessory

4.7.8 Screen 8: Successful Adding of an Accessory

After a successful bootstrapping attempt, the user can either proceed back to the *Control Panel* with the newly added accessory through the *Next* Button. Otherwise, cancel the process using *Cancel* Button and return to the *Control Panel* but without any new accessory added.

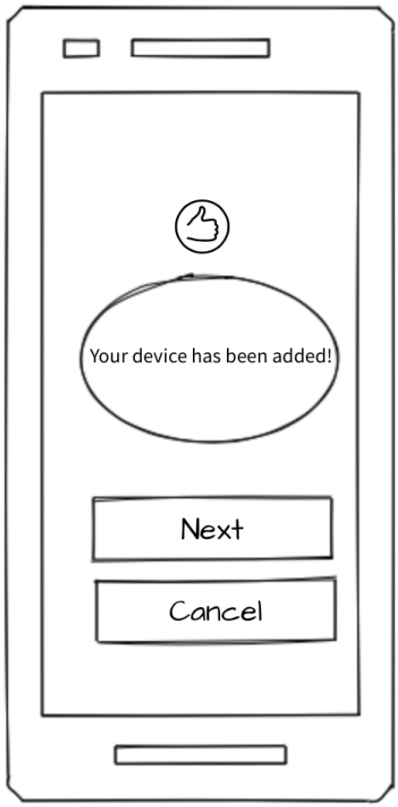

Figure 4.16: Successful Attempt to Add Accessory

5 Discussion and Analysis

Secure management and operation of IoT devices in people's homes is one of the key issues in consumer IoT. This thesis investigates an approach to secure management and operation of IoT devices in people's homes by adapting the LwM2M industrial standard (LwM2M [4]) and using the smart phone as the Controller of home IoT devices. A system architecture based on OMA LwM2M specifications [4] is designed; an Android Application is implemented as a proof of concept. Chapter 5 discusses and analyses the findings with regards to the underlying technologies used to implement the solution i.e, LwM2M; and securing credentials and managing data and secure usability that requires users intervening in the bootstrapping process.

In the rest of Chapter 5 the research questions stated in Section 1.2 are discussed. For the reader's convenience, the questions are repeated:

- RQ1: How to securely manage the life cycle of the device using a Controller device?
- RQ2: What types of data should be transferred between the Controller and the simple device and how to securely transfer and store them safely during bootstrapping (initial setup) and operational modes (after bootstrapping)?
- RQ3: How can the Controller protect long-term credentials that need to be provisioned to the simple device against cyber attacks?
- RQ4: How to provide a user friendly experience during the bootstrapping of

the device using QR code reading?

5.1 Securing Credentials and Managing Data

During the device manufacturing, certain security measures (e.g data generated using high entropy random function) are injected into the device securely in the factory premises. The security measures may include one or all the following: the device's serial number, Pre-shared Key (PSK) and/or X.509 Certificate.

In this thesis, the scenario of using the Certificate is taken into consideration for device authentication in OOB channels. Certificate authentication is based on Public key infrastructure. Therefore the simple device with 32 bit MCUs must support one of the following digital signature algorithms (RSA, DSA, ECDSA). ECDSA is used for the purpose of being a better candidate for IoT devices in terms of energy consumption and data throughput, specifically the curve secp256r1 outperformed the curve secp224r1 [40].

To elaborate more on the answer for RQ1, The X.509 Certificate injected into the simple IoT device which contains the public key that is used to authenticate the device to the Controller during the bootstrap process when the User launches the HomeApp on the Controller (mobile phone) and attempts to scan the QR code containing a hash of the device's certificate (128 bits). The QR code message structure supported WiFi and Bluetooth but for the demonstration of the proof of concept of the thesis, only WiFi in-band channel is used as illustrated in depth in Section 4.3.

Based on industrial-scale IoT devices, the LwM2M Clients (IoT Devices) are pre-configured in a way that they can contact the LwM2M Bootstrap Server to begin the bootstrapping process. Configuring the IoT devices to contact a centralized Bootstrapping entity that is outside the consumer's local network can lead to the following privacy issues: (1) the centralized LwM2M Bootstrap Server requires the knowledge of the LwM2M Server (Controller) which is typically a consumer device

such as a smart phone to complete the bootstrapping. (2) When the LwM2M Client contact the Bootstrap Server, the centralized server can build a profile of consumers. Therefore, for more protection, a Controller that communicates with a simple device is placed in a local network. In is important to note that, the LwM2M Client (simple device) cannot be pre-configured with Bootstrap Server Account information in a home context. This raises a security issue as if this were to be true, any Controller connected to the local network will be able to influence the simple device to bootstrap with it. To reduce the mis-binding possibility, private network details (simple device's access point) are exhibited by the simple device to the Controller. The access point of the simple device is based on PKI, an advantage for this is that a database is not needed to keep the information of all the trusted peers in the simple device, since a peer can be verified from a certificate presented when it attempts to connect to the simple device. In this way it acts as a CA.

Moving to the Controller, the general role of the controller lies in the life cycle management of a device that starts from bootstrapping the device followed by managing the operations of the device and performing the necessary updates to the device such as updating the system components and firmware. A more specific role of the controller is securing the communication in OOB channel and in in-band channel. An important argument is the "creation" of trust when using public keys or shared keys in OOB channel.

A key component in android security is providing endpoint authentication using X.509 certificates. In this thesis self-signed certificates are used instead of CAs. The certificate verified for it's hash using the public key retrieved from the QR code encoded data. The LwM2M Server certificate is ignored on the LwM2M Client side. The significance of using channel binding is when tampering with the content of the QR code or the LwM2M Client certificate, i.e. such as in MiTM attacks. This will immediately invalidate the certificate, therefore protecting the security of the

communication channel. Another point is that the Android Key store ensures that the keys stored do not enter the application process so that even if the application process is compromised the intruder cannot gain access of the keys.

For RQ3, During the bootstrapping procedure, the operational credentials that are explained in Section 4.4 are provisioned to the LwM2M Client. These credentials are transferred securely in a DTLS layer. The credentials include the key pair, Bootstrap Server Certificate, connectivity settings which will be used by the device when restarting in operational mode to connect to the home network like the Controller in order to send and receive requests. The connectivity settings are stored in an encrypted file format using Encrypted Shared Preferences as specified in Section 4.6. To decrease the scope of possible exploitation, the permission requests in the application are limited to those strictly needed to avoid any mishandling of any of the sensitive data associated with the permission as discussed in detail in Section 2.6.2.

When it comes to securing management data such as security, wireless and light objects (since the example scenario is controlling the LED). It is important to secure the communication between the device and the Controller which is achieved using an encrypted channel i.e. DTLS. Upon sending the HTTP requests to the server and then the CoAP messages to the device a DTLS connection allows for protection of the management data against cyber attacks. The management data does not have to be stored locally in the Controller device, even if it is the more secure option. As that limits the chances for an attacker attempting to intercept the communication tunnel to gain access of the management data in case it is stored in a server locally in a home gateway or over a smart hub such as Amazon Echo[1]. The options of storing the management data are also referred to as different setup configurations in Section 4.2. However, data privacy challenges exist that come with cloud storage.

[1]Amazon Echo: https://www.amazon.com/smart-home-devices

Some of the requirements include: (1) encryption of files with high-entropy keys prior to uploading the files on the cloud. (2) The decryption keys are required to only be accessible by the devices within the network domain [41]. From a secure usability perspective, in addition to these security requirements, it is important that the device authorization remains minimal and consistent.

5.2 Usable Security

In order to address RQ4, "Usable Security" is about involving the end user in the process and providing a user friendly interface without compromising the security of the system. Users should be limited to the parts of the system that they have access to, to grant users authorization to certain tasks as they assist in critical processes.

In this prototype for instance, the bootstrapping cannot occur without the support of the user in terms of launching the app then choosing the pairing method of preference (in this case QR code reading) and making the effort to point the mobile device to the correct QR code. This is so a successful bootstrapping can occur.

Users hold the responsibility to correctly scan legitimate QR codes; as it is quite common for malicious QR codes to contain compromised code that directs users to secondary sites that contain possible risks on the security and privacy of the devices. In addition to tampering with QR codes, users can be prone to phishing attacks or cross-site scripting attacks [15]. QR codes can include more complex schemes to make the process more difficult for the attacker to modify the original QR code.

According to a research by Kainda, Flechais, and Roscoe [13] a number of factors affect the security, usability and eventually the success of OOB channels from a user's prospective. Users vary in the level of motivation to perform the security tasks depending on the circumstances. Users can be distracted while performing these tasks which leads to them shifting their attention away from the pairing process. During OOB channel achievement, the application must not be demanding of the

user's constant attention which might lead to frustration on the user's behalf in most scenarios. In fact, this creates bigger chances of security failures. This in addition to the reasons discussed in the previous paragraph, has contributed to choosing a visual and effective means of bootstrapping to avoid impatient user reaction during a crucial authentication step.

Other secondary methods of bootstrapping are using OCR and manual entry which are left to the user to decide which to use. Manual entry can be frustrating depending on the length of the URL to be entered. According to the same study [13], Compare and Confirm[2] and Compare and Select[3] methods have been ruled out due to being highly responsible for security failures. This is where the choice of the manual entry emerged to decrease the frustration of the user while entering a string of values. The *Checksum* value in the URL helps identify the correctness of the string during entry and alerts the user meanwhile.

Another aspect of usability enhancement that was taken into consideration when designing this application that is not security related is attempting to reduce the user's wait perception through better design practices. According to a study on The Effects of Visual Feedback Designs on Long Wait Time [42] results have shown that there existed a correlation between the user's perceived speed and the wait evaluations. This translates to better wait experiences when progress speed is estimated to be faster. The use of the progress bar is applied during attempting to bootstrap the device as shown in Figure 5.2. These finding have played a major part in understanding the cues needed to manipulate the user's wait experience to maintain user's attention which is estimated to be about 10 seconds [43]. Findings in 2012 by Lallemand and Gronier [44], confirmed that progress bars are common indicators for

[2]CC: One way authentication where the user is required to press a button on their device only. The hash value representations vary from numeric to alphanumeric and sound.

[3]CS: A string is displayed on one of the devices and four strings of the same format are displayed on the other device. Values only include numeric and alphanumeric

interface feedback and that users preferred non-linear speeds and preferred shorter times unlike longer ones.

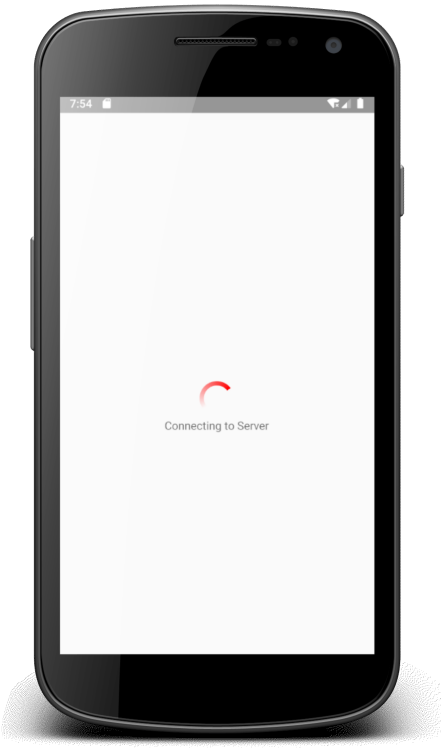

Figure 5.1: Use of progress bar during bootstrapping attempt

5.3 Limitations

Thie thesis was produced in the times of a widespread pandemic (COVID-19) and with governmental regulations that were imposed to limit the spread of the virus through restricting movement and enforcing social distancing regulations. Such circumstances represented additional challenges to testing the prototype on a range of devices and users. Based on these conditions, in Section 5.2 no data was provided to support the claims of this thesis regarding the smoothness and user-friendliness nature of the app and the appeal to the users.

In addition to having an effect on testing the usability of the application, recommendations following working remotely placed limitations on testing the different components of the system proposed. This had a direct impact on increasing the time span for testing separate units and the communication needed between the team members, which directly reduced the efficiency of the work.

5.4 Future Work

Implementation wise and regarding the different fields of the OOB message in Section 4.3, it would be beneficial to support more in-band channel communication protocols in addition to WiFi and other device credential types in addition to using X.509 certificates such as using Raw Public Key (RPK) or Pre-Shared Key (PSK).

The next step of the framework would be to support other interfaces in addition to Bootstrapping and Device Discovery and Registration that are mentioned in Section 4.2. The support should be extended to Information Reporting where the LwM2M Server reports periodically to an LwM2M Client certain resources when such a criteria are matched such as exceeding a specific threshold. In addition to that, allowing the controller to be able to authenticate and control multiple devices (nodes) at once. When there are many devices to bootstrap to, it becomes very tidious to bootstrap with each and every device so it would easier to bootstrap them all at once [45].

Another useful addition to the thesis, would be to implement Multi Dynamic Host Configuration Protocol (MDHCP) on the LwM2M Client to be able to dynamically allocate the multicast of addresses of the devices connected to the same access point. This would reduce power consumption as the device shouldn't be on all the time and it decreases response time as the device shouldn't wait a long time before using a multicast address.

Finally, since the framework has been implemented over two different parts that

is the mobile device and the PC, the next step would be to move the Servlet classes from the Server to Android locally.

6 Conclusion

Secure management and operation of IoT devices in people's homes is one of the key issues in consumer IoT. In this thesis an approach has been investigated where this issue is solved by using a mobile device as a Controller device. A solution for securely managing and using IoT devices in a home context has been designed and implemented. This solution is based on OMA Lightweight Machine to Machine (LwM2M) specifications [4].

The initial setup of the IoT device is done when the user scans the QR code of the device with his/her smartphone's camera. The security of the device's setup is based on the information in that QR code, which could include, e.g., the hash of the device's public key. Other methods to do the device's setup using smartphone have been considered as well. After the initial setup, the IoT device joins the home WiFi network and is controlled from the smartphone of the user.

As part of a proof-of-concept demonstration an Android application has been implemented for a user's smartphone that does the QR code reading. The *HomeApp* utilizes Android Key Store to secure the generation and storage of keys, and uses DTLS to secure communication between the devices. The application provides a smooth user interface for connecting and controlling the IoT device. The implementation of the IoT device part in this proof-of-concept demonstration is a topic of another master's thesis [37].

Based on this implementation work it can be concluded that the approach inves-

tigated in this thesis is practical, and it would be worthwhile to continue working in this direction in the future. Future work could also include a usability study that could not be done due to restrictions caused by the COVID-19 pandemic.

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (IoT): A vision, architectural elements, and future directions", *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, Sep. 2013.

[2] M. G., N. Kushalnagar, H. J., and C. D., "Transmission of IPv6 packets over IEEE 802.15.4 networks", *IETF*, no. RFC4944, Jun. 2007, Standard.

[3] G. Alberto and Y. F. Hu, "Wireless sensor networks: A survey on the state of the art and the 802.15.4 and zigbee standards", *Computer Communications*, vol. 30, no. 7, pp. 1655–1695, May 2007.

[4] OMA SpecWorks. (2018). Lightweight M2M (LWM2M). Technical Specification, [Online]. Available: `http://www.openmobilealliance.org/release/LightweightM2M/V1_1-20180710-A/OMA-TS-LightweightM2M_Core-V1_1-20180710-A.pdf` (visited on 02/18/2020).

[5] L. F. Rahman, T. Ozcelebi, and J. Lukkien, "Understanding IoT systems: A life cycle approach", *Procedia Computer Science*, vol. 130, pp. 1057–1062, May 2018.

[6] O. Garcia-Morchon, S. Kumar S.and Keoh, R. Hummen, and R. Struik, "Security challenges in the IP-based internet of things", *Wireless Personal Communications*, vol. 61, pp. 527–542, Dec. 2011.

[7] D. Garcia-Carrillo and R. Marin-Lopez, "Lightweight CoAP-based bootstrapping service for the internet of things", *Sensors*, vol. 16, no. 3, p. 358, Mar. 2016.

[8] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz, "Extensible authentication protocol (EAP)", *IETF*, no. RFC3748, Aug. 2008, Standard.

[9] T. Ylonen and E. C. Lonvick, "The Secure Shell (SSH) protocol architecture", *IETF*, no. RFC4253, Jan. 2006, Standard.

[10] F. Stajano and R. Anderson, "The resurrecting duckling: Security issues for Ad-hoc wireless networks", *Lecture Notes in Computer Science (LNCS)*, vol. 1796, pp. 172–182, Sep. 1999.

[11] F. Stajano, R. Anderson, and J.-H. Lee, "Security policy", in *Advances in Computer Science*, vol. 55, United States of America: Elsevier, Jul. 2001, pp. 185–231.

[12] S. Latvala, M. Sethi, and T. Aura, "Evaluation of out-of-band channels for IoT security", *SN Computer Science*, vol. 1, pp. 1–20, Sep. 2019.

[13] R. Kainda, I. Flechais, and A. W. Roscoe, "Usability and security of out-of-band channels in secure device pairing protocols", in *Proceedings of the 5th Symposium on Usable Privacy and Security - SOUPS '09*, California, United States of America: ACM Press, Jul. 2009, pp. 1–12.

[14] C. Gehrmann, C. J. Mitchell, R. Holloway, S. Tw, and K. Nyberg, "Manual authentication for wireless devices", vol. 7, no. 1, pp. 29–37, Apr. 2004.

[15] K. Krombholz, P. Fruhwirt, P. Kieseberg, I. Kapsalis, M. Huber, and E. Weippl, "Qr code security: A survey of attacks and challenges for usable security", in *Human Aspects of Information Security, Privacy, and Trust*, Crete, Greece, Jun. 2014, pp. 2–8.

[16] Y.-W. Kao, G.-H. Luo, H.-T. Lin, Y.-K. Huang, and S.-M. Yuan, "Physical access control based on qr code", in *Proceedings of the 2011 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, ser. CYBERC '11, United States of America: IEEE Computer Society, Oct. 2011, pp. 285–288.

[17] M. T. Goodrich, M. Sirivianos, J. Solis, G. Tsudik, and E. Uzun, "Loud and clear: Human-verifiable authentication based on audio", in *26th IEEE International Conference on Distributed Computing Systems*, vol. (ICDCS'06), Lisboa, Portugal, Jul. 2006, pp. 10–16.

[18] N. Karapanos, C. Marforio, C. Soriente, and S. Capkun, "Sound-proof: Usable two-factor authentication based on ambient sound", in *24th USENIX Security Symposium (USENIX Security 15)*, vol. 24, Washington, D.C.: USENIX Association, Aug. 2015, pp. 483–498.

[19] N. A. Chattha, "NFC - vulnerabilities and defense", in *2014 Conference on Information Assurance and Cyber Security (CIACS)*, vol. 24, Rawalpindi, Pakistan: IEEE, Jun. 2014, pp. 35–38.

[20] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (CoAP)", *IETF*, no. RFC7252, pp. 5–12, Jun. 2014, Standard.

[21] N. Modadugu and E. Rescorla, *The design and implementation of datagram TLS*, Report, 2004.

[22] H. Tschofenig and T. Fossati, "Transport layer security (TLS) / datagram transport layer security (DTLS) profiles for the internet of things", *IETF*, no. RFC7925, Jul. 2016, Standard.

[23] Apple. (2020). About Apple HomeKit, [Online]. Available: `https://developer.apple.com/homekit/` (visited on 02/17/2020).

[24] Eve. (2020). Eve Energy Smart Plug and Power Meter, [Online]. Available: https://www.evehome.com/en/eve-energy (visited on 02/17/2020).

[25] Apple. (2020). MFi Program, [Online]. Available: https://developer.apple.com/programs/mfi (visited on 02/17/2020).

[26] Apple Developer. (2017). Homekit accessory protocol specification. Non-Commercial Version, [Online]. Available: https://developer.apple.com/support/homekit-accessory-protocol (visited on 03/18/2020).

[27] D. Taylor, T. Wu, N. Mavrogiannopoulos, and T. Perrin, "Using the secure remote password (SRP) protocol for tls authentication", *IETF*, no. RFC5054, Nov. 2007, Standard.

[28] Huawei Security Protection Technology Lab, "Lifecyle management of simple devices (unpublished internal document)", Draft White Paper, Mar. 2020.

[29] S. Jiang. (Jan. 2017). China's xiaomi targets 2017 sales of 14.5 billion usd after 2016 business overhaul, [Online]. Available: https://www.reuters.com/article/xiaomi-outlook/chinas-xiaomi-targets-2017-sales-of-14-5-bln-after-2016-business-overhaul-idUSL4N1F22BA (visited on 03/13/2020).

[30] J. Tour. (May 2018). Xiaomi global community, [Online]. Available: https://xiaomi-mi.com/news-and-actions/xiaomi-the-worlds-largest-iot-platform/ (visited on 03/20/2020).

[31] D. Giese, "Reversing IoT: Xiaomi ecosystem", presented at the DEF CON 26, Zurich, Switzerland, Aug. 2018. [Online]. Available: https://hitcon.org/2018/CMT/slide-files/d2_s1_r0.pdf (visited on 04/08/2020).

[32] Google. (2020). Device security, Cloud IoT Core documentation, and Google Cloud, [Online]. Available: https://cloud.google.com/iot/docs/concepts/device-security (visited on 04/08/2020).

[33] Android. (2020). Android Keystore System, [Online]. Available: `https://developer.android.com/training/articles/keystore` (visited on 05/08/2020).

[34] R. v. Rijswijk-Deij and E. Poll, "Using trusted execution environments in two-factor authentication: Comparing approaches", in *Open Identity Summit 2013*, vol. P-223, At Kloster-Banz, Germany: Gesellschaft für Informatik e.V., Sep. 2013, pp. 20–31.

[35] A. Vasudevan, E. Owusu, Z. Zhou, J. Newsome, and J. McCune, "Trustworthy execution on mobile devices: What security properties can my mobile platform give me?", vol. 7344, Jun. 2012, pp. 159–178.

[36] Z. Fang, W. Han, and Y. Li, "Permission based android security: Issues and countermeasures", *Computers Security*, vol. 43, no. C, pp. 205–218, Jun. 2014.

[37] T. Akgün, "Secure life cycle management of iot devices)", To be published Msc. Thesis, University of Alto, Master's thesis, Jul. 2020.

[38] Leshan. (2020). Leshan Wiki, [Online]. Available: `https://github.com/eclipse/leshan/wiki` (visited on 02/18/2020).

[39] Mock Flow. (2020). MFi Program, [Online]. Available: `https://www.mockflow.com` (visited on 03/18/2020).

[40] M. Suárez-Albela, T. M. Fernández-Caramés, P. Fraga-Lamas, and L. Castedo, "A practical performance comparison of ECC and RSA for resource-constrained IoT devices", *2018 Global Internet of Things Summit (GIoTS)*, vol. 18, pp. 1–6, Jun. 2018.

[41] A. Paverd, S. Tamrakar, H. L. Nguyen, P. K. Pendyala, T. D. Nguyen, E. Stobert, T. Gröndahl, N. Asokan, and A.-R. Sadeghi, "Omnishare: Securely accessing encrypted cloud storage from multiple authorized devices", vol. abs/1511.02119, pp. 1–9, Dec. 2016.

[42] S. Li and C.-H. Chen, "The effects of visual feedback designs on long wait time of mobile application user interface", *Interacting with Computers*, vol. 31, no. 1, pp. 1–12, Mar. 2019.

[43] R. B. Miller, "Response time in man-computer conversational transactions", in *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, ser. AFIPS '68 (Fall, part I), San Francisco, California: Association for Computing Machinery, Dec. 1968, pp. 267–277.

[44] C. Lallemand and G. Gronier, "Enhancing user experience during waiting time in hci: Contributions of cognitive psychology", in *Proceedings of the Designing Interactive Systems Conference, DIS '12*, vol. DIS '12, NewCastle, United Kingdom: Association for Computing Machinery, Inc, Aug. 2012, pp. 751–760.

[45] E. E. Journal. (2014). Probme simplifies thing wifi connection, [Online]. Available: `https://www.eejournal.com/2014/09/16/probme-simplifies-thing-wifi-connection/` (visited on 06/20/2020).

[46] B. Sarikaya, M. Sethi, and D. Garcia-Carrillo, "Secure IoT bootstrapping: A survey", *IETF*, Sep. 2018.

[47] M. Weiser, R. Gold, and J. S. Brown, "The origins of ubiquitous computing research at PARC in the late 1980s", *IBM Systems Journal*, vol. 38, no. 4, pp. 693–696, Aug. 1999.

[48] W. R. Clay and D. Shin, "Secure device pairing using audio", in *43rd Annual 2009 International Carnahan Conference on Security Technology*, Zurich, Switzerland: IEEE, Oct. 2009, pp. 77–84.

[49] J. M. McCune, A. Perrig, and M. K. Reiter, "Seeing-is-believing: Using camera phones for human-verifiable authentication", *IEEE Symposium on Security and Privacy*, vol. 4, p. 23, May 2005.

[50] N. Saxena, J.-E. Ekberg, K. Kostiainen, and N. Asokan, "Secure device pairing based on a visual channel", in *2006 IEEE Symposium on Security and Privacy (S&P'06)*, vol. 6, California, United States of America: IEEE, Dec. 2006, pp. 6–313.