Constrained IoT authentication in Cloud services

Tomás Ribeiro da Silva

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisor: Prof. Ricardo Jorge Fernandes Chaves

Examination Committee

November 2021

Dedicated to my girlfriend Rita, whose unconditional love and support kept me going.

Acknowledgments

I want to thank my family and friends, for their support and patience, that was crucial during my time at University.
I also want to thank Professor Ricardo Chaves and Carlos Morgado who assisted me during the course of this work. A big thank you to Truphone for giving me the opportunity to develop this work, with a special thanks to Truphone's R&D team who welcomed me into their team as one of their own.

Resumo

Os dispositivos restritos que operam em redes limitadas representam uma parte importante do paradigma IoT. Estes dispositivos são utilizados em diferentes indústrias como: agricultura [1], metereologia [2] e gestão de cadeias de logística [3]. Com o surgimento de serviços de computação e armazenamento na Nuvem e tecnologias IoT, é imperativa a criação de um mecanismo transparente de autenticação capaz de identificar estes dispositivos a outras entidades. Uma solução compacta para este problema é aqui explorada, ao alavancar a autenticação da rede celular para identificar e autenticar estes dispositivos restritos que operam em redes limitadas para entidades externas. Este trabalho expõe um protocolo de autenticação transparente para o cliente, especificamente desenhado para dispositivos restritos que operam em redes limitadas. Isto é alcançado com a utilização de um protocolo baseado no OAuth2.0 que corre em cima de protocolos da camada de aplicação melhorados e com a introdução de um novo participante no protocolo. Nesta nova abordagem ao problema, o novo componente, o *"broker"*, permite libertar o cliente de todo o processo de autenticação e permite a delegação de todas as tarefas relacionadas com o mesmo, como os pedidos e gestão dos tokens de acesso. Quando comparado com o mecanismo de autenticação atualmente utilzado pela Truphone, a solução proposta reduz o tráfego de rede no lado do cliente em mais de 98%, reduz o seu custo computacional e pode ser usada sobre redes NB-IoT, a rede celular que dará conectividade a estes dispositvos restritos.

Keywords: IoT, Autenticação, Dispostivos restritos, Redes Celulares

Abstract

Constrained devices operating on constrained networks are a big part of the Internet of Things paradigm. They have a myriad of applications such as farming [1], wheater-monitoring [2] and logistics and supply-chain management [3]. With Cloud Services and IoT technologies on the rise, it is imperative to create a seamless authentication mechanism that can identify these devices to a third-party Cloud provider. A lightweight solution to this problem is explored, by leveraging network-level authentication via SIM to identify and authenticate these constrained devices operating on constrained networks to a third-party Cloud provider. This work presents an authentication protocol that is transparent to the client, specifically designed for constrained devices operating on constrained networks. This is achieved using an OAuth 2.0-based protocol running on top of improved communication and application-layer protocols, with the introduction of a new key feature, the *"broker"*. In this novel approach, the broker component allows the client to be free from all authentication procedures and allows the delegation of all of these tasks, like managing and requesting access tokens. When compared to the authentication mechanisms currently working in Truphone, the proposed solution reduces the client-side network traffic by more than 98%, reduces computation needs and is suitable to use over NB-IoT, the cellular network that will connect these constrained devices.

Keywords: IoT, Authentication, Constrained devices, Cellular Networks

Contents

List of Tables

List of Figures

Nomenclature

3GPP 3rd Generation Partnership Project

5G Fifht Generation

ACE Authentication and Authorization for Constrained Environments

AES CCM Advanced Encryption Standard-Counter with cipher block chaining message authentication

AES-CBC-MAC AES Message Authentication Code

AES GCM Advanced Encryption Standard in Galois/Counter Mode

API Application Programming Interface

AuC Authentication Center

AWS Amazon Web Services

CA Certificate Authority

CBOR Concise Binary Object Representation

CoAP Constrained Application Protocol

COSE CBOR Object Signing and Encryption

CPU Central Processing Unit

CWT CBOR Web Token

DTLS Datagram Transport Layer Security

ECDSA Elliptic Curve Digital Signature Algorithm

EdDSAEdwards-Curve Digital Signature Algorithms

eUICC Embedded Universal Integrated Circuit Card

GSM/2G Global System for Mobile Communications/Second Generation

HMAC Hash-Based Message Authentication Code

HSS Home Subscriber Service

HTTP HyperText Transfer Protocol

HTTPS HyperText Transfer Protocol Secure

IBM International Business Machines Corporation

ICC-ID/SSN Integrated Circuit Card ID/SIM Serial Number

IMSI International Mobile Subscriber Identity

IoT Internt of Things

IP Internet Protocol

iUICCIntegrated Universal Integrated Circuit Card

JSON JavaScript Object Notation

JWT JSON Web Token

LTE/4G	Long Term Evolution/Fourth Generation
M2M	Machine to Machine
MAC	Message Authentication Code
MIME	Multipurpose Internet Mail Extensions
MQTT	Message Queue Telemetry Transport
MQTTS	Message Queue Telemetry Transport Secure
MQTT-SN	Message Queue Telemetry Transport - Sensor Networks
NB-IoT	Narrowband IoT
NIDD	Non-IP Data Delivery
OSCORE	Object Security for Constrained RESTful Environments
OSI	Open Systems Interconnection
QoS	Quality of Service
RAM	Random Access Memory
RFID	Radio-Frequency Identification
ROM	Read Only Memory
RTT	Round-trip Time
SDK	Software Development Kit
SIM	Subscriber Identity Module
SMTP	Simple Mail Transfer Protocol
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Data Protocol
UICC	Universal Integrated Circuit Card
UMTS/3G	Universal Mobile Telecommunications System/Third Generation
USIM	Universal Subscriber Identity Module

Chapter 1

Introduction

This chapter introduces the proposed work by stating its motivation, providing an overview of the topic and presenting the main objectives to be fulfilled by the proposed solution. A brief thesis outline concludes this chapter.

1.1 Motivation

The motivation of this work is to leverage the existent network-based authentication to trustfully and automatically authenticate a device to a third-party service, such as a Cloud provider (e.g. AWS IoT Core). Since the network provider can already authenticate and trust a device, by extending this trust with a mechanism to authenticate the device to a third-party, the same trustful and automatic authentication can be achieved. Constrained devices operating on constrained networks currently do not have out-of-the-box solutions for lightweight authentication to a third-party. Since constrained devices and constrained networks play an important role in a large scope of IoT (Internet of Things) scenarios, this is an industry shortcoming that this work attempts to solve.

1.2 Overview

Most authentication mechanisms that allow authentication for third-party services are not optimized for the IoT paradigm, targeting regular computers, this is a different scenario from the one that will be explored due to the limitations of IoT devices. On the other hand, most authentication mechanisms designed specifically for IoT are made with an entirely different objective, double way authentication between two IoT devices, and not authenticating the IoT device to a third-party service.

1.3 Objectives

The objective of this work is to provide an autonomous authentication mechanism for constrained IoT devices operating on constrained networks that allows the device to authenticate itself to a third-party service by leveraging the network-based authentication mechanism already provided by cellular networks. The main requirements for this work, as defined in conjunction with Truphone, are:

1. To authenticate a device based on his cellular connection;
2. Reduce the device-side traffic network by more than 50% when compared to current solutions;
3. Build a seamless and transparent authentication mechanism for the device;
4. Reduce the computation on the device-side.

1.4 Thesis Outline

This thesis is divided into seven chapters, including the introduction. The second chapter provides the necessary context and information to understand the rest of this work. The third chapter describes other solutions for similar problems. The fourth chapter details the proposed solution, how it works and intermediate solutions that have been developed. The fifth chapter presents some technical implementation details. The sixth chapter details the tests made to the proposed solution and discusses the obtained results. Finally, the seventh and last chapter concludes this work by stating its achievements and some future work direction.

Chapter 2

Background

This chapter covers the background needed for this work. It starts by detailing the role of the SIM Card in cellular communications and the evolution achieved in SIM card technology in the recent past. It then introduces the reader to some generic cryptographic algorithms, their inner workings and the difference between symmetric and asymmetric cryptography followed by an introduction to the Internet of Things. It then proceeds to the OSI model and the description of its layers.
The rest of the chapter is dedicated to specific protocols essential to this work, namely: UDP, TCP and their differences, use-cases and shortages; TLS and its session control mechanisms; NB-IoT and how this protocol provides the ideal connectivity to constrained devices operating on constrained environments; CoAP and MQTT, two widely used transport-layer protocols for IoT devices; JSON and JWTs; and finalizes with CBOR and CWTs.

2.1 SIM Card

For the past 30 years, the Subscriber Identity Module (SIM) card has been responsible for providing user authentication and secure communication in mobile networks. It provides a seamless and secure method for user authentication. What is widely known today as a SIM card is, in most cases, a Universal Integrated Circuit Card (UICC) running SIM and Universal Subscriber Identity Module (USIM) applications. The SIM application is necessary to connect to Global System for Mobile Communications/Second Generation (GSM/2G) networks, while the USIM application is used to connect to Universal Mobile Telecommunications System/Third Generation (UMTS/3G), Long Term Evolution/Fourth Generation (LTE/4G) and Fifth Generation (5G) networks. This distinction will be made throughout this work, and from now on, the physical card will be referred to as UICC.

More recent advances in technology have expanded the way UICCs can operate. The Embedded UICC (eUICC) is a non-removable chip that allows for remote SIM provisioning, which eliminates the need for a removable card. This will be explored in-depth in this section. The natural evolution in UICC technology is the Integrated UICC (iUICC), that takes UICC integration to the next level by incorporating the eUICC chip with an application chip, giving it additional cost and area benefits. This solution will be detailed later in this section.

Smart Cards

Smart cards are plastic cards with an embedded chip that allow the card to store data and perform computations[4]. Apart from storing data, a smart card provides security by protecting against unauthorized access and data manipulation. These cards have the capability of performing cryptographic operations in the card to protect the data. Data can be stored inside a smart card and never accessed from outside sources. This type of device can then be used to store private keys that should never be revealed to other parties.

UICC Architecture

The UICC [5] is a smart card, which means that it provides the features mentioned above. The UICC is considered to be tamper-resistant. The UICC has its own RAM, ROM, CPU and storage area; the power and clock are provided from the outside; and it can be considered as a secure environment.
In mobile networks, the card is the root for user authentication. The SIM/USIM software loaded in the UICC contains data that is essential to authentication and user identification, namely:

- The Integrated Circuit Card ID/SIM Serial Number (ICC-ID/SSN) which is a 19-20 digit number that uniquely identifies a UICC.
- The International Mobile Subscriber Identity (IMSI) which is a 15 digit number that uniquely identifies a mobile subscriber.
- An authentication key that is shared with the provider.

The UICC can also perform authentication algorithms that allow the authentication key to never leave the card.
Most UICCs cards also have a Java-Card based architecture, depicted in Figure 2.1, which runs a separate core operating system from the device's operating system. This separation, in combination with the secure interfaces present in the Java-Card architecture, creates an extra layer of security in the communication between the UICC and the device, essentially allowing

the card to act as a hardware firewall between the mobile device and the information on the card.

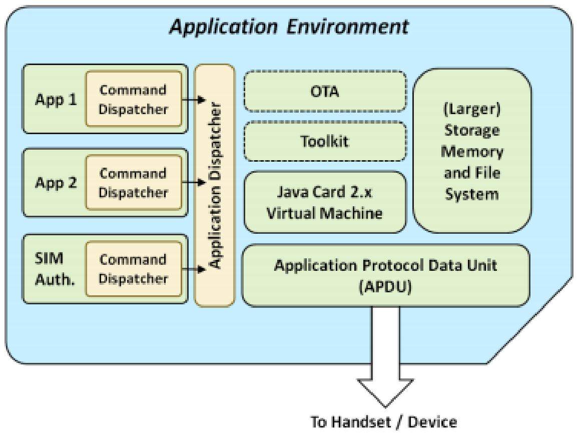

Figure 2.1: Current Java Card SIM Architecture [5]

UICC Authentication

In every mobile network generation, since GSM, two main components can be identified, the UICC and the Authentication Center (AuC) which differs from network type. However, it always serves the same function, storing the authentication key that is mirrored in the UICC and providing the user authentication in the network. This shows the importance of the UICC since it allows the user to be authenticated through the storage of the authentication key.

eUICC

The eUICC [6], as stated above, is the embedded version of the UICC. This new level of integration brings forth new opportunities. In traditional UICCs the SIM/USIM applications, that contain the authorization keys used for authentication in mobile networks, are bound to the physical card. This means that each physical card has a different identity, and if a user wants to change his provider, he must change the UICC for a new one. Since the eUICC is embedded in the user's device and cannot be changed, there is an obvious need for a new solution that can support these non-removable cards.
This spawned the innovative solution of Remote SIM Provisioning. The eUICC is capable of storing multiple SIM profiles. Each SIM profile contains the information that is available on a traditional removable UICC, including the IMSI and the authorization key. These profiles are downloaded from the operator, and the way this profile managing is achieved depends on whether the eUICC is working with the consumer solution or the machine-to-machine (M2M) solution. In the consumer solution, this is managed by a human which is the end-user. The

end-user can actively choose their network provider and require a high level of user interaction. In the M2M solution, this is not needed, as it targets IoT devices and it allows for a large number of devices to be managed remotely without user interaction.
The M2M solution presents the most interesting scenarios allowing for an easy, scalable and seamless way of managing a big number of IoT devices. The consumer solution also provides some interesting features like the tracing of stolen devices via the mobile network (given that the eUICC cannot be removed in contrast to a traditional UICC) and, since the eUICC is smaller than the UICC, it frees space in the device which is always useful when dealing with smartphones, tablets or other types of consumer electronics.

iUICC

The iUICC [7] is an integrated version of the UICC, meaning that the capabilities of the UICC are not in a separate chip like in the eUICC solution, but directly integrated within the application chip. This reduces component count, simplifies board integration, and enables resource sharing within the chip. This prevents the duplication of resources, such as a memory controller and peripherals for the UICC, for the application processor chip, and the modem module, since these resources can be shared between them.
As seen in Section 2.1, the UICC is the basis for the security and authentication in mobile networks, so the iUICC must stay secure even when some resources are shared with other chips in the device. The manufacturer must provide a trusted secure environment in which the SIM/USIM applications can run. These applications must be logically and physically shielded from other components. This chip must also have cryptographic capabilities to implement the mobile protocols used in mobile networks and it should be able to run the remote SIM provisioning protocols as the iUICC, being non-removable, just like the eUICC, will need to use them. It should also be able to run Java Card applications as traditional UICCs are. To summarize, the iUICC should not lose any functionality that is present in the traditional UICC and it should only create new advantages, like the reduced cost and space occupied in the board.

2.2 Cryptography

To understand this work it is essential to grasp basic cryptographic algorithms. Namely the difference between encryption, signature and hashing algorithms and the difference between symmetric and asymmetric algorithms.
An encryption algorithm is used to encrypt a message and protect it against unrequested access providing confidentiality to the message. It works in the following way: we have two users,

Alice and Bob, who need to exchange secret messages over a public network and Eve, another participant, with access to the public network. Alice and Bob have previously met and share a secret key. Alice then writes a message, called a plaintext, then uses the secret shared key and an encryption algorithm to encrypt the message, now called a ciphertext, and publishes it to the public network. Bob will then receive the ciphertext and uses the shared secret key and the encryption algorithm to decrypt the ciphertext back to a readable plaintext. Eve also has access to the ciphertext, however, she cannot decrypt it. Figure 2.2 depicts this flow.

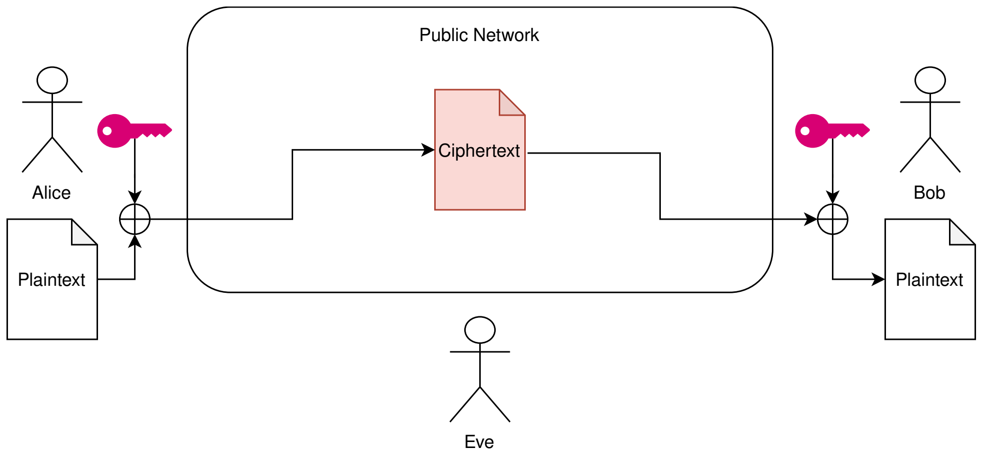

Figure 2.2: Generic Encryption Algorithm

A hashing algorithm can be used to produce a Message Authentication Code (MAC) providing integrity to the message. A hash function is a one-way function that receives data and a secret key and produces a hash (also called MAC). Since two different messages produce two different hashes, a hash can be used to verify the integrity of the data. In this example, Alice needs to send a message to Bob over a public network and Bob needs to be certain that the message was not tampered with in any way. In this example there is a new participant, Mallory, who cannot only read the messages exchanged over the public network but also capture them, re-write them and publish her own messages. Alice and Bob share a secret key in this example. Alice starts by writing the message and then uses the shared secret key to hash this message, attaches the hash to the message and then publishes it. Bob will then receive the message and use the secret shared key to hash it. If his hash matches the one in the message then Bob can be certain that the message was not tampered with. If Mallory re-writes the message the hash will not match and Bob can discard the message. This example is depicted in Figure 2.3.

This example depicts a basic hashing protocol, that is not safe to use due to length extension attacks. In a length extension attack Mallory, could append information to the message and produce a valid hash if she knew the initial hash and message and the length of the key, the first two are public information and the last Mallory could brute-force. To deal with this problem,

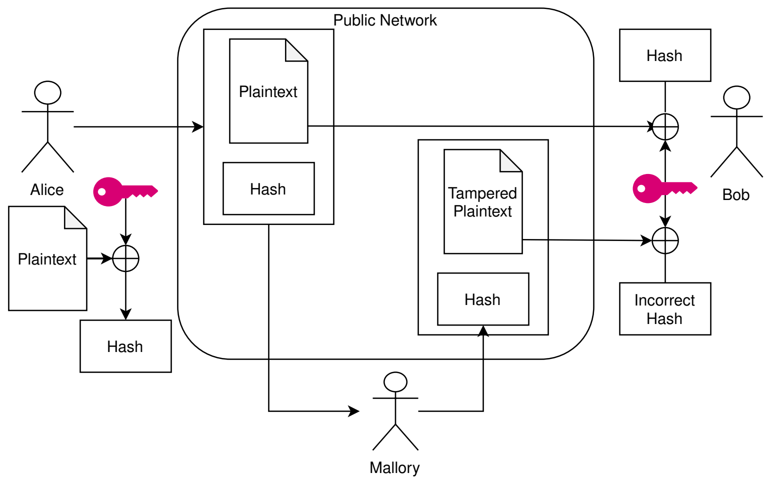

Figure 2.3: Generic Hashing Algorithm

Alice should use an Hash-Based Message Authentication Code HMAC. To use HMAC Alice starts by writing the message and then uses the shared secret key to derive two keys: the inner key and the outer key. Then Alice hashes the message along with the inner key to produce hash 1 and then hashes hash 1 with the outer key to produce hash 2. Alice then attaches hash 2 to the message and publishes it. Bob will then receive the message and use the secret shared key to do the same process. If his hash matches the one in the message then Bob can be certain that the message was not tampered with. If Mallory re-writes the message the hash will not match and Bob can discard the message, and the message is also safe against the aforementioned length extension attack.
Signature algorithms can be used to provide authentication, integrity and non-repudiation to messages. In this example, Alice and Bob do not share a secret key. Instead, Alice has a pair of keys: a private and a public one. A pair of keys can be used to encrypt and decrypt a ciphertext. The private key must be kept a secret by Alice and the public key can be known by everyone. Bob has Alice's public key. Now Alice needs to send a message to Bob over a public network and Bob needs to be certain that the message was not tampered with and that Alice wrote the message. Mallory is present again to try and sabotage this message exchange. Alice starts by writing the plaintext, then she uses her private key to encrypt the message and produce a ciphertext. She then attaches the ciphertext to the plaintext and publishes it. Bob will then use Alice's public key to decrypt the ciphertext and, if the decrypted ciphertext matches the plaintext then Bob can be certain that the message was not tampered with and that it comes from Alice. If Mallory tries to alter the plaintext, then the decrypted ciphertext produced by Bob will not match the plaintext and, if she tries to replace the message with a message written by her, Bob will not be able to decrypt the ciphertext with Alice's public key as depicted in

Figure 2.4.

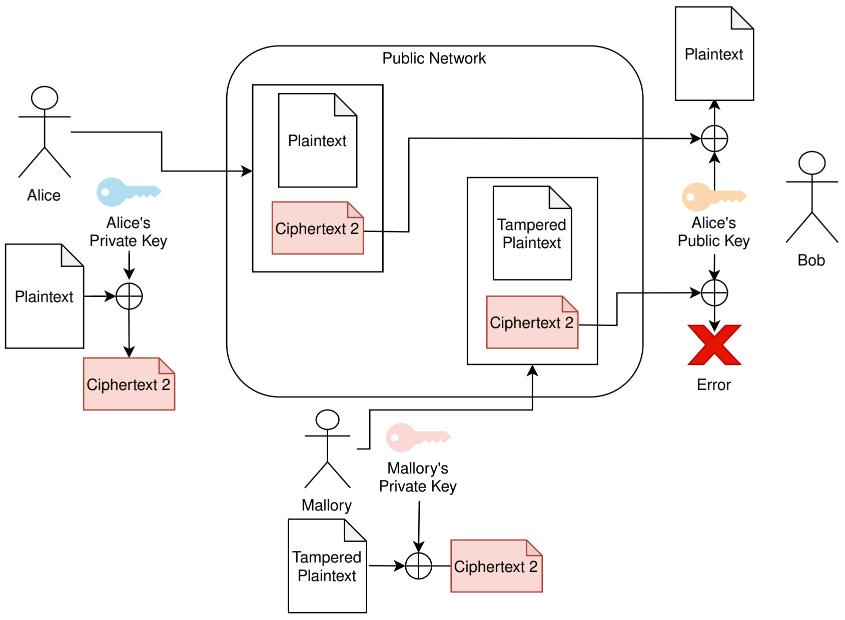

Figure 2.4: Generic Signing Algorithm

Symmetric and asymmetric algorithms are differentiated by the type of keys used. When two parties only use one key to communicate then the algorithm is symmetric. When two parties use a pair of keys to communicate then the algorithm is asymmetric. Examples in Figures 2.2 and 2.3 depict symmetric algorithms and in Figure 2.4 the example uses an asymmetric algorithm.

Trust

Another important concept in cryptography is trust. An entity is said to be trusted if it can be assumed to always behave according to a given protocol. It is assumed that a trusted entity does not collude with an attacker or that the trusted entity is itself an attacker.

In the Internet there are some entities that are considered trusted third-parties and they are the root for authentication in the web. These entities are Certificate Authorities (CA) and along with asymmetric encryption they are responsible for assuring that web-servers are who they claim to be. A CA is responsible for assuring the identity of web-servers and it issues a certificate that the web-server will use to prove its identity. This certificate contains the web-server identity, its public key and a digital signature from the CA. When a client connects to a web-server the web-server will send it it's public key and certificate. The client can then verify the server's identity by checking the certificate. Browsers come preloaded with information from CAs that allow the client to verify its digital signature.

2.3 Internet of Things

The Internet of Things is the term used to describe the new technology paradigm envisioned as a global network of machines and devices capable of interacting with each other [8]. With a myriad of applications in enterprise cases, IoT is an important part of the so-called Industry 4.0 [9]. With an estimated growth from $7.6 * 10^9$ units in 2019 to $24.1 * 10^9$ units installed all over the world in 2030 and estimated revenue of $150 * 10^6$ US dollars [10] the IoT market will have a big impact soon.

The Internet of Things is characterized by the heterogeneity of technologies, devices and protocols being used which makes it hard to define key characteristics.

Special attention will be devoted to constrained IoT devices and networks. In some cases, the devices have special constraints. Some devices are designed to have a battery life of 10+ years which limits the power consumption by applications, while others have limited computational capabilities and certain machines operate on lossy networks or can only transmit a limited number of bytes per day due to network limitations. These devices are a special case as when developing IoT solutions their limitations must be taken into consideration. As seen in [11], there is a lot of potentials for the IoT industry to grow with numerous applications, and an important note from this review is that most technologies detailed here deal with constrained devices or constrained networks, as can be seen by the listing of Radio-Frequency Identification (RFID) and sensor networks.

2.4 OSI Model

Open Systems Interconnection (OSI) model is a conceptual framework that uses seven layers to describe how applications running upon network-aware devices communicate [12][13]. In the OSI model, each layer runs on top of the other, as depicted in Table 2.1. This means that each layer provides service to each other, with layer N providing services to N+1 and receiving service from N-1. There are different protocols for each layer and each one fulfils a different objective in the model. Their purpose will be explained in the following subsections.

Physical Layer

The physical layer defines the physical and electrical characteristics of the network. It is responsible for the exchange of information at a physical layer using electrical, radio or optical signals. Some well-known standards in this layer are Ethernet and Bluetooth.

Layer 7	Application
Layer 6	Presentation
Layer 5	Session
Layer 4	Transport
Layer 3	Network
Layer 2	Data Link
Layer 1	Physical

Table 2.1: OSI Model

Data-Link Layer

The data-link layer provides reliable transmission of data between adjacent nodes, while also performing error detection and controlling to the physical layer. Bridges and switches operate on this layer.

Network Layer

The network layer organizes data for transfer and reassembly in the data-link layer. In short, the main function of this layer is path determination and logical addressing. The Internet Protocol (IP) operates in this layer.

Transport Layer

The transport layer provides reliable transmission of data between two machines operating on different networks. It also performs error detection and control to the network layer. The Transmission Control Protocol (TCP) and the User Data Protocol (UDP) operate on this layer.

Session Layer

The session layer, as the name implies, deals with session management, namely session termination, recovery, suspension and restart. The Transport Layer Security (TLS) Protocol operates on this layer.

Presentation Layer

The presentation layer deals with unpacking and packing the data sent by the lower layers to the application layer. It exists to provide independence between the end-user and the way the data is sent from one machine to another. The Multipurpose Internet Mail Extensions (MIME) protocol is a good example of a protocol running on this layer that serves the Simple Mail Transfer Protocol (SMTP) running in the application layer.

Application Layer

The application layer is the layer that interacts directly with the end-user. Another good example of a protocol running in this layer is the Hypertext Transfer Protocol (HTTP).

2.5 UDP and TCP

UDP and TCP are two widely used transport layer protocols [14]. These two protocols are very distinct and it is important to understand their differences in both reliability and performance. TCP is a connection-oriented protocol based on positive acknowledgements [15] that guarantees a reliable connection. This means that TCP can recover from packet loss and duplication. Since TCP is a connection-oriented protocol, this means that there is the need to establish and terminate a session. Session establishment is made through the use of a three-way handshake and session termination is made using another three-way handshake [16].

UDP, unlike TCP, provides connectionless service. This means that each user datagram sent by UDP is an independent datagram. There is no reliability features or error recovery and packet loss can occur. Since UDP is connectionless, there is no need for session establishment or termination protocols.

TCP offers a more reliable service than UDP at the cost of more overhead. When the most important aspect of an application is performance, UDP should be used. However, it is important to note that the packet loss that can occur with UDP makes the use of this protocol unsuitable for applications that need reliability.

2.6 TLS

TLS is one of the most widely used cryptographic protocol for secure communications on the Internet [17]. It replaces the use of the Secure Sockets Layer Protocol (SSL) [18]. It is a stateful, connection-oriented, client-server protocol providing authentication, integrity, and confidentiality for two parties [19]. Session establishment is made through the use of the TLS handshake. The handshake begins when the client reaches out to the server with a "client hello" message. This message serves mainly to initiate the communication but it also provides additional technical information, such as the protocol versions and cypher suites supported by the client. The server then responds to this message by sending three different messages: a "server hello" message which defines the session identifier and the chosen protocol version and cypher suite to be used; a "certificate" message that contains the server's certificate and a "server hello done" message that marks the end of this stream of server messages. Now, the client will authentic-

ate the server's identity by using the certificate. This certificate is signed by a trusted CA, a trusted entity that is responsible for ensuring the correct identity of the parties. By using the CA's certificate (which usually is preloaded in the client), the client can validate the server's certificate and prove that he is who he claims to be. In the remaining of the handshake, both parties agree on a key to secure their communication channel using a symmetric key protocol.
In the most recent version for this protocol, TLS 1.3, a new handshake is presented which, by condensing some of the messages, results in a faster handshake. The 0-Round-Trip Time (RTT) Data mode is also introduced [20]. This mode allows for encrypted data to be sent by the client appended in the "client hello" message, called early data. It can only be done if a session was established earlier and this is the resumption of a session and not the beginning of a new one or if the client and the server have a pre-shared key. This has a trade-off as the early data is not secured against replay attacks which means that, if the attacker gets hold of this data, they can send it repeatedly to the server and the server cannot detect that this data does not come from the client.

DTLS

Datagram Transport Layer Security (DTLS) is a similar protocol to TLS, designed to protect datagram-based applications. While TLS runs over TCP, DTLS runs over UDP.
As TLS, DTLS provides authentication, integrity, and confidentiality for two parties. DTLS reuses almost all the protocol elements of TLS, with minor but important modifications for it to work properly with datagram transport. TLS depends on a subset of TCP features: reliable, in-order packet delivery and replay detection [21]. Since DTLS is made to work over stateless protocols messages might not be delivered. To deal with this DTLS implements a retransmission mechanism in which both endpoints have a timer and keep retransmitting the last message sent until a reply is received.

2.7 NB-IoT

Narrowband IoT (NB-IoT) is a radio access technology, introduced by the 3rd Generation Partnership Project (3GPP) to provide a low-cost, low-power, wide-area cellular connectivity for the Internet of Things [22]. NB-IoT was designed with the following objectives [23]: improve indoor coverage; support a massive number of low throughput devices; reduce complexity; limit latency to ten seconds; co-exist with legacy systems and minimize impact in the base stations hardware. To fulfill this last objective, NB-IoT uses the same networks elements that LTE uses. It only needs some minor adaptations to the Home Subscriber Service (HSS) in order for an already existing LTE network to support NB-IoT. Nonetheless, using devices connected by NB-IoT presents some other challenges: the devices send very small payloads; are usually in sleep mode for most of the time; have infrequent data transactions; operate on networks with narrow bandwidth and high latency and transmit an average of 200 bytes per day [24]. These limitations are challenging as they impose restrictions on the protocols that can be used for communication. A layered view of the technologies used in constrained IoT scenarios is depicted in Figure 2.5. It is possible to see that NB-IoT acts as the basis of a constrained IoT model and that it can transfer data using either UDP or Non-IP Data Delivery (NIDD) protocols.

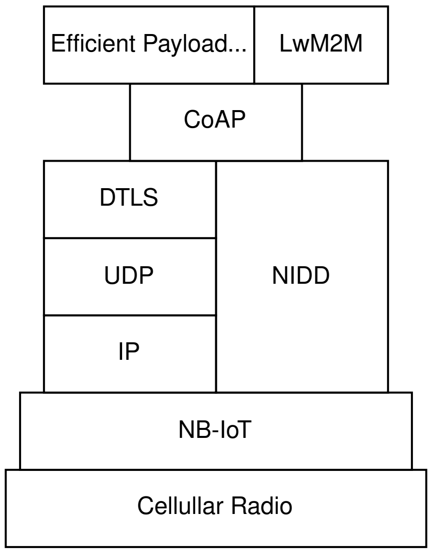

Figure 2.5: Constrained IoT Model [24]

2.8 CoAP

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained networks [25]. It works similarly to HTTP in client-server mode. It provides interesting features that were made to deal with the challenges of constrained devices and networks like low header overhead, asynchronous message exchanges and the ability to run over UDP or NIDD protocols.

As UDP is inherently unreliable, CoAP defines two types of messages, namely confirmable messages and non-confirmable messages to define its own reliability mechanism. The former requires an acknowledgement similar to the mechanism used in TCP communications while the latter does not require any kind of acknowledgment [26].

As described in [24], CoAP offers better performance when using it over NIDD. There is, of course, a trade-off here given that, by not using UDP, CoAP loses access to the security features

assured by the use of Datagram Transport Layer Security (DTLS). DTLS is identical to TLS but running over UDP instead of TCP.
It is important to note that while DTLS is not supported when using NIDD protocols, this choice has security features. NB-IoT networks (as well as cellular networks from and including UMTS/3G) provide the air link encryption and ensure the protection of the data transmitted via the 3GPP Cellular IoT architecture in the operator domain [27].

2.9 MQTT

MQTT (Message Queuing Telemetry Transport) is a standardized publish-subscribe protocol that was released by the International Business Machines Corporation (IBM) in 1999 [28]. It uses a topic-based publish-subscribe architecture. This means that when a client publishes a message M to a particular topic T, then all the clients subscribed to the topic T will receive the message M [29]. MQTT defines three types of participants in the protocol: publishers, subscribers and brokers. Publishers and subscribers can both be considered “clients” as they are connected to a broker (often called the server) that manages the correct distribution of the messages. Publishers connect to the broker and publish messages to a designated topic and all clients that are connected to this broker and subscribed to the said topic will receive it. The clients are never connected directly to each other and only to the broker.
MQTT runs over TCP and it offers 3 Quality-of-Service (QoS) levels. The first QoS, QoS 0, works in an at-most-once fashion. The message is sent only one time without any guarantee that it will arrive. There is no acknowledgement of any kind and the message will either be received one time or not at all. It is the lowest quality of service but it is also the lightest as there is no overhead from any message confirmation mechanisms. The second QoS, QoS 1, works in an at-least-once fashion. The message is assured to be received at least one time but it can be sent and received multiple times. Every message sent is acknowledged and assured to be received. The acknowledgement messages bring more overhead to this QoS when compared to the QoS 0. The final QoS, QoS 2, works in an exactly-once fashion. The message is assured to be received exactly one time. This is assured by a four-way handshake between the client and the broker. While this guarantees that the message is received only one time, it also comes with significantly more overhead than QoS 1 and 0. MQTT does not define any security mechanism to encrypt or protect messages. This means that in order to secure communications made over MQTT there is the need to use secure underlying protocols. TLS is used for this purpose, and like HTTP, when MQTT is running on top of TLS it is mentioned as MQTTS.

MQTT-SN

MQTT-SN [30] is an adaption of MQTT designed for sensor networks. Since devices operating on sensor networks are extremely constrained devices, regular MQTT is not an option for them. As stated in Section 2.9, MQTT runs over TCP, which is a costly protocol, introducing a sizeable overhead. This overhead is normally not a big deal, however, sensor networks do not operate in normal conditions and, with the restrictions on the devices, this overhead will be intolerable. To solve this issue MQTT-SN runs over UDP instead of TCP.
MQTT-SN was designed to work over any network service, that provides a bi-directional data transfer service between any node and a particular one (a gateway). MQTT-SN was made with the regular MQTT in mind, and it offers a good way of having a single machine serving both MQTT-SN and MQTT requests.

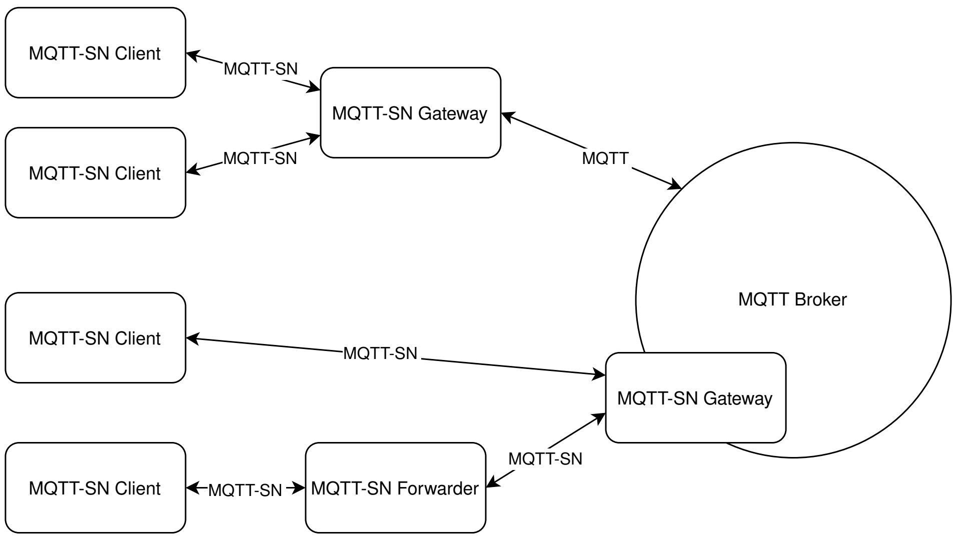

Figure 2.6: MQTT-SN Architecture

As seen in Figure 2.6, this protocol has 3 key components: the MQTT-SN clients, the MQTT-SN forwarders and the MQTT-SN gateways. The MQTT-SN clients are the constrained nodes. The MQTT-SN gateways have the role of translating from MQTT-SN to MQTT to communicate with the main MQTT broker (which is the main server). In cases where the main server is not serving MQTT requests, the gateways are not necessary since there is no need to translate from MQTT-SN to MQTT. The MQTT forwarders, as the name says, are only present in the architecture to forward messages from the clients to the gateways when there are no direct connections between them. The main feature that MQTT-SN provides is that it maintains the structure of MQTT but is made to work with constrained devices.

2.10 JSON

JavaScript Object Notation (JSON) [31] is a standard by the Internet Engineering Task Force (IETF) that describes a lightweight, text-based, language-independent data interchange format. It was derived from the ECMAScript Programming Language Standard. JSON defines a small set of formatting rules for the portable representation of structured data. JSON is a text format for the serialization of structured data and can represent four primitive types (strings, numbers, booleans, and null) and two structured types (objects and arrays).
Nowadays, a considerable number of web applications provide an external API (Application Programming Interface) consisting of a set of JSON-based services. More than 40% of the APIs included in ProgrammableWeb return JSON data, where all services are interrelated [32].

JWT

JSON Web Token (JWT) [33] is an internet standard for two parties to securely transfer the information as a JSON object. It is a token format that is usually used for information exchange and authorization. A JWT is used for a party to send information about a subject to another party, who then will use this information to provide or decline access to a protected resource. The information about the subject comes in the form of claims. These claims represent assertions that the authorization server makes about the identity of the client. JWTs can be signed or MACed to prove the authenticity or the integrity of the JWT. As per [34] an access token is a JWT that follows the following requirements:

1. The JWT contains an "iss" claim, that stands for "issuer"and represents the entity that issued the token.

2. The JWT contains a "sub" claim, that stands for "subject" and represents the entity that is identified by the token.

3. The JWT contains an "aud" claim, that stands for "audience" and represents the authorization server as the intended audience for the token.

4. The JWT contains an "exp" claim, that stands for "expiration time" that limits the time window during which the JWT can be used

5. The JWT is either MACed or signed by the issuer.

There are more claims that a JWT may contain but they are not mandatory. One of them is the "jti" which stands for "JWT ID" and this claim contains a unique identifier for the token. This claim is especially important as it plays the role of mitigating replay attacks. The authorization server can store the “jti” values used for the length of time for which the JWT is valid, expressed in the “exp” claim. It is relevant to note that the protocol does not specify any key distribution method to enforce requirement 5. leaving the implementation to the responsibility of the developer.

2.11 CBOR

The Concise Binary Object Representation (CBOR) [35] is another standard by the IETF that describes a data format, similar to the JSON but designed to include the possibility of extremely small code size and fairly small message size. It is a binary serialization data format that aims to be coded with a small code footprint, small message size, and extensibility with no need for negotiating the version [36].

CWT

CBOR Web Token (CWT) [37] is a compact means of representing claims to be transferred between two parties. The claims in a CWT are encoded in the CBOR, and CBOR Object Signing and Encryption (COSE) is used for added application-layer security protection. CWT is derived from JWT but uses CBOR rather than JSON.
A CWT does not have any mandatory claims, however, it must have a valid COSE header [38]. This header defines whether the CWT is signed, MACed or encrypted and the cryptographic algorithm used. If any special parameter needs to be sent to validate or decrypt the CWT (like an initialization vector), it must be sent in this header. The header must not be included in the signed, encrypted or MACed message and its only purpose is to provide the relevant information for a receiving party to be able to decrypt the CWT.
COSE allows the usage of two distinct signing algorithms, two MAC algorithms and three encryption algorithms. A signed CWT can be made using either the Elliptic Curve Digital Signature Algorithm (ECDSA) or the Edwards-Curve Digital Signature Algorithms (EdDSA). To MAC a CWT either Hash-Based Message Authentication Codes (HMACs) or the AES Message Authentication Code (AES-CBC-MAC) should be used. When encrypting a CWT one should use either the Advanced Encryption Standard in Galois/Counter Mode (AES GCM), the Advanced Encryption Standard-Counter with cypher block chaining message authentication (AES CCM) or ChaCha20 and Poly1305 combined.

2.12 Overview

This chapter introduced the necessary knowledge to tackle the problem that this work seeks to solve. It covers the evolution of SIM Cards, cellular technology and the architecture and capabilities of the SIM. It exposes the reader to the IoT paradigm by covering its limitations and the heterogeneity in technologies used in this paradigm. It also provides a brief introduction to cryptography and the OSI Model which are two concepts that are heavily used throughout this work. The rest of the chapter covers network protocols and data formats that are relevant not only for this dissertation but when discussing IoT, cellular networks or authentication mechanisms in general.

Chapter 3

State of the Art

This chapter covers the state of the art related to the problem at hand. Section 3.1 covers the OAuth 2.0 authorization framework and Section 3.2 covers the ACE framework, which was designed specifically for constrained devices operating in constrained environments. Section 3.3 introduces the existing solution at Truphone.

3.1 OAuth2.0

OAuth 2.0 [39] is an authorization framework that allows users to authenticate towards a remote resource/service. This framework defines four participants: the client, the resource owner, the resource server and the authorization server. The **client** is the entity that requests access to a protected resource on behalf of the resource owner, and it can only access the protected resource once it has been granted authorization. The **resource owner** is the entity that can grant permission to the client to access said resource. The **resource server** is the entity hosting the protected resource, allowing access to the resource when presented with an access token. The **authorization server** is the entity capable of issuing access tokens after successfully authenticating the resource owner and obtaining permission.

3.1.1 Protocol Flow

The interactions between these roles are depicted in Figure 3.1. **(A)** marks the start of the interaction between all parties, with a request for authorization to access the protected resource, made by the client to the resource owner. **(B)** depicts the response to the authorization request, where the resource owner grants the client authorization to access the protected resource. In **(C)**, the client presents this authorization grant to the authorization server which, after authenticating the client and validating the authorization grant, issues an access token that is sent to the client

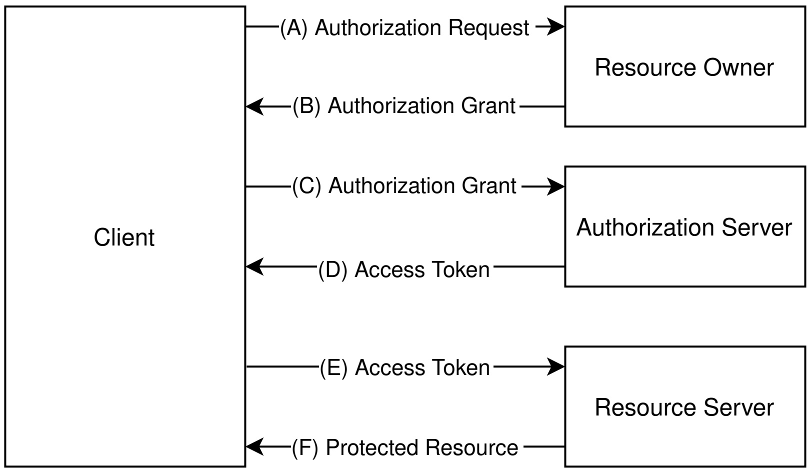

Figure 3.1: OAuth 2.0 Protocol

in **(D)**. In **(E)**, the client requests the protected resource to the resource server, sending the access token in the request. The resource server validates the access token and responds to the client by allowing access to the protected resource in **(F)**. An important caveat is that the client works on behalf of the resource owner, meaning that steps **(A)** and **(B)** are usually not actual message exchanges between different entities but the a end-user, acting as the resource owner, using an application, acting as the client, to access information that he controls on another service/application.

3.1.2 Confidential Clients and Public Clients

OAuth 2.0 makes an important distinction regarding clients. Clients can either be public or confidential. **Confidential** clients can hold credentials in a secure way without exposing them to unauthorized parties. **Public** clients cannot. So confidential clients are, for example, web applications running in a secure server and public clients are, for example, single-page applications running directly on a browser.

3.1.3 Resource Owner Password Credentials Flow

The resource owner password credentials flow is designed for cases when the resource owner has a trust relationship with the client. In this flow, the client has access to the resource owner credentials. This flow is only used when other flows are not viable and it can be used to migrate clients to OAuth 2.0 by converting the stored resource owner credentials into access tokens. This flow is depicted in Figure 3.2.

The protocol starts in **(A)**, here the resource owner provides the client with its credentials. The way these credentials are shared is out of the scope of the protocol, so, as long as the client

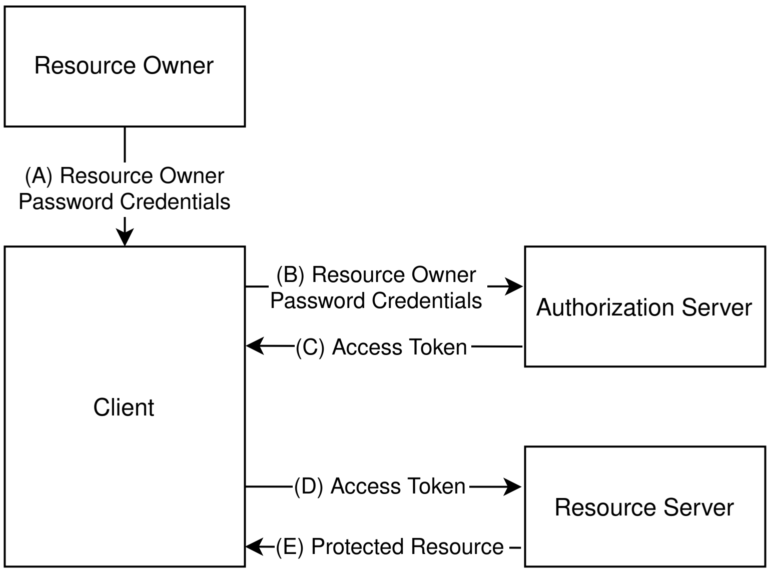

Figure 3.2: Resource Owner Password Credentials Flow

can access the resource owner's credential, step **(A)** is completed. In **(B)** the client requests an access token from the authorization server and sends the resource owner credentials along with this request. After validating the resource owner's credentials, the authorization server issues an access token that is sent to the client in **(C)**. In **(D)**, the client uses this access token to request access to a protected resource held by the resource server. After validating the access token, the resource server grants access to the protected resource, depicted in the Figure as **(E)**.

3.1.4 Client Credentials Flow

The client credentials flow is used when the client is requesting access to resources owned by itself or when it is accessing resources under the control of another resource owner. This resource owner has previously orchestrated, with the authorization server, that the client can access it without contacting the resource owner (the way this orchestration is performed is not defined by OAuth 2.0 and it is considered out of scope of the standard). In these cases, there is no need to contact the resource owner and the client can access the protected resource by authenticating itself to the authorization server. In this flow, the client directly contacts the authorization server and requests the access token. This flow is illustrated in Figure 3.3.
In **(A)** we can see that the client authenticates itself to the authorization server (this authentication can be done by any method; a common one is to use a username and a password to identify a client). The authorization server, after authenticating the client, issues an access token which is represented in **(B)**. **(C)** and **(D)** are the same steps shown in Figure 3.1 where the client sends the access token to the resource server and the resource server allows the client to access the protected resource.

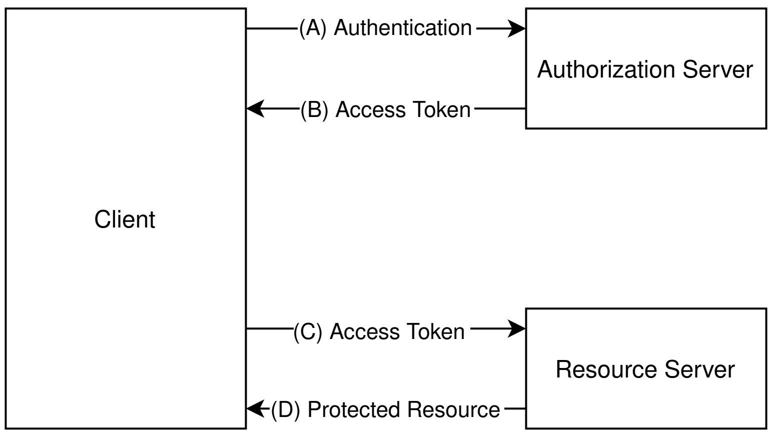

Figure 3.3: Client Credentials Flow

3.1.5 Access Token

The access token is the token that proves that the client has the authorization to access the protected resource. This access token has the form of a JWT, described in Section 2.10. The access token to be considered valid must be a valid JWT and the resource server must be able to validate this token. This validation includes validating the signature and the "aud" claim which must indicate that the token was issued to access the determined protected resource.

3.2 ACE Framework

The Authentication and Authorization for Constrained Environments (ACE) Framework [40] is a framework designed to expand the authorization flow of OAuth 2.0 to constrained devices in IoT networks. One key difference that aims to accommodate the limitations of constrained devices and networks is the use of CWTs instead of JWTs. By using CBOR, the protocol can better compress the messages, resulting in smaller payloads.

The ACE protocol, illustrated in Figure 3.4, works similarly to OAuth 2.0. It starts when the client (the constrained device) asks the Authorization Server for an access token that contains claims about the identity of the client and if the client can access the protected resource here depicted in **(1)** and **(2)**. Then, the client sends this access token in the request to access the resource hosted by the Resource Server, as shown in **(3)**. The Resource Server, if needed, can verify the token with the Authorization Server using an introspection request. This introspection request is portrayed in steps **(4)** and **(5)**. The resource server sends the Access Token, that the client used, to the Authorization Server. The Authorization Server then confirms the validity of the Access Token. After validating the token (with or without an introspection), the Resource Server provides the client with access to the protected resource, as shown in **(6)**. This protocol was made to work over CoAP however, HTTP implementations are possible.

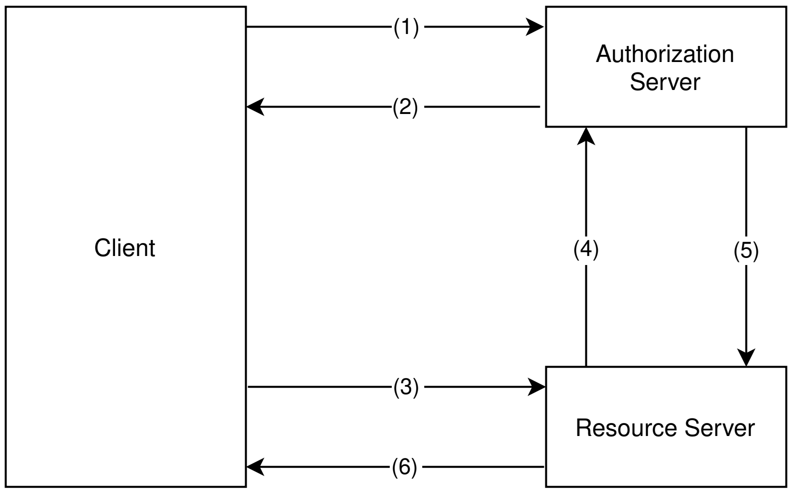

Figure 3.4: ACE Protocol Flow

The implementation in [41] is made over HTTP due to the poor support of DTLS by the networking libraries implementing CoAP servers and clients showed poor support for DTLS at the time. This implementation showed that, by using CWT instead of JWT, the access tokens were about 50% smaller. The reported overhead due to the use of HTTP is around 50%, which, when dealing with constrained devices and networks, is a very large number. This implementation is extremely useful in showing the massive difference in performance that is possible by using CWT instead of JWT.
This protocol was made to work with two different security protocols, one working with Object Security for Constrained RESTful Environments (OSCORE) [42] and another working only with DTLS. OSCORE allows for true end-to-end encryption (over proxies) while DTLS does not, since CoAP defines a number of proxy operations that require transport layer security to be terminated at the proxy. It is important to notice that OSCORE runs on the application layer and DTLS on the transport layer so they are not mutually exclusive.

3.3 Truphone's Existing Solution

Truphone has a patent pending working solution (patent ID PCT/GB/2021/051093) capable of authenticating IoT devices equipped with a Truphone SIM to a third-party. It is based on the Client Credentials Flow from OAuth2.0 described in Section 3.1.4. This solution, also known as the "notarizer", is here described starting with the participants in the protocol and then detailing the flow.

Truphone's Private Network

This solution leverages the authentication and security present in cellular networks. An eSIM equipped device, using LTE/NB-IoT, has a secure channel to the core cellular network. The device is connected to the core network which then connects to the Internet. This core network/secure channel will be named Truphone's Private Network. The authorization server in Truphone's existing solution is deployed in this core network, where it has access to a mapping between a certain client's IP and their SIM credentials. This is caused by the authentication process in cellular networks. A client that has access to the Internet through a cellular network is authenticated beforehand via SIM. The cellular provider is responsible for IP address provisioning and it can associate this IP to a certain SIM card. By leveraging this association, the authorization server knows the identity of every client that is connected to Truphone's network.

3.3.1 Protocol Participants

As mentioned in the previous section, the "notarizer" is based on OAuth 2.0 and so they share some participants, namely the authorization server and the client. The client, as in OAuth 2.0, is the participant that needs to be authenticated, the authorization server is the entity that has the ability to identify and authenticate the client and, in OAuth 2.0, there is the resource owner who needs to assess the identity of the client and in the "notarizer" this role is filled by the third-party, which needs to assess the identity of the client. These participants will now be detailed.

Authorization Server

The authorization server has the same responsibilities as its namesake in the ACE framework and OAuth 2.0. It is responsible for identifying clients and issuing access tokens that assert a client's identity. This authorization server however has a major difference from the ones presented in the aforementioned protocols. It is capable of authenticating every client that issues a request without the need for credentials to be exchanged. The authorization server is a service provided by Truphone and it is deployed in Truphone's network. As seen in Section 3.3, this means that the authorization server has a secure channel established to the client and can identify it.

Client

The client is the entity that needs to be authenticated to another entity by the authorization server (in this solution to the third-party service). In this proposed solution, the client is a constrained IoT device operating in a constrained environment, with a Truphone SIM. The client also has a third-party Software Development Kit (SDK) pre-installed which provides the client with the necessary information to contact the third-party service, namely the IP address. The client is connected to the Internet using a Truphone eSIM. This connection allows the client to communicate with Truphone's services using a private network.

Third-Party Service

The third-party service is the equivalent of the resource owner in the ACE framework and OAuth 2.0. It expects clients to request access to some protected resource and it relies on the authorization server for the identification of these clients. Working with the authorization server can provide, or deny, access to this resource. In this solution, due to third-party constraints, the client uses MQTTS (with one-way authentication, so the third-party authenticates itself to the client, which has a preloaded certificate to confirm the third-party identity) to communicate with the third-party.

3.3.2 Protocol Flow

Truphone's existing solution's flow is depicted in Figure 3.5.

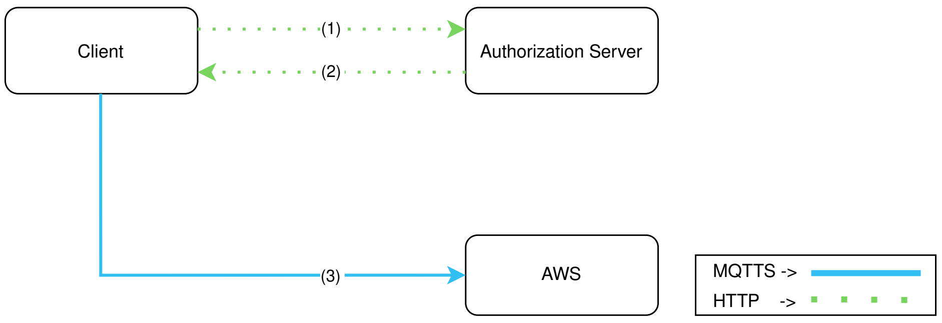

Figure 3.5: Notarizer

In **(1)**, the client, through an HTTP request, asks for a token to the authorization server. In this request, the client will send to the authorization server SIM credentials (namely two unique identifiers: ICC-ID and IMSI). Since the authorization server is deployed in Truphone's private network, it has access to the mapping between a certain client's IP address and their SIM credentials, allowing the authorization server to identify the client. In **(2)**, the authorization server responds to this request with the token issued for the client, and finally, in **(3)**, the client uses this token to authenticate itself towards the third-party service using MQTTS. An important detail, is that while the connection between the Client and the Authorization Server is not protected by HTTPS, it is a private connection, running on Truphone's private network. This flow is triggered when the client needs to publish a message to a third-party.

3.3.3 Access Token

The token follows the specification defined in the OAuth 2.0 standard. It is a JWT containing the mandatory OAuth 2.0 claims, described in Section 3.1.5, signed by the authorization server. Token validation is done by the third-party. Token validation also follows the OAuth 2.0 standard. A key-sharing mechanism between the authorization server and the third-party is not defined as it is considered out-of-scope for this project.

3.3.4 Limitations and Advantages

The “notarizer” is not the best solution when dealing with constrained devices for a myriad of reasons. Firstly, the use of HTTP is a problem. HTTP runs on top of TCP which, as discussed in Section 2.5, provides a connection-oriented reliable service at the cost of additional overhead when compared with UDP. The use of a JWT poses another problem. As seen in Section 2.11, there are other alternatives that can encode the same information as a JWT while saving space, namely a CWT which trades off human readability for compactness. Another problem is the use of MQTTS. As described in Section 2.9, MQTTS is a version of MQTT which runs on top of TLS. The use of MQTT by itself could already be considered problematic considering that it runs on top of TCP, however, the use of TLS makes this even worse. The TLS handshake presents an unacceptable overhead when dealing with constrained devices.

3.3.5 Security

To prove the security of Truphone’s existing solution, an assumption was made: OAuth 2.0 is a secure protocol, designed for use on the Internet over HTTPS. The security of the intermediate solutions will be demonstrated by addressing the changes it has compared to OAuth 2.0.
The “notarizer” implements the following changes compared to OAuth 2.0:

1. The use of HTTP instead of HTTPS, when requesting the token
2. The use of MQTTS instead of HTTPS, when sending the token to authenticate itself

Both are changes to application-layer protocols. The second change is meaningless as HTTPS security comes from the use of TLS and MQTTS also runs over TLS. This means that both HTTPS and MQTTS provide the same security capabilities. The first change, however, can pose problems as HTTP is not secured by TLS. Nonetheless, this connection between the client and the authorization server is, as mentioned before, made over Truphone’s Private Network that guarantees authentication and confidentiality, keeping the protocol secure.

Chapter 4

Proposed Solution

This chapter presents and details the proposed solution. First, the problem description is restated. The building blocks of the proposed solution are then introduced followed by the details of the proposed solution. The chapter finishes with the major improvements that this work brings when compared to the existing solution detailed in the previous chapter, namely the introduction of a new protocol participant, the broker that performs most of the authentication protocol on behalf of the client.

4.1 Problem Description

This work aims to solve the lack of an automatic authentication mechanism for constrained IoT devices operating in constrained environments that allows devices to authenticate themselves towards third-party services. The proposed solution targets constrained devices that, due to operating in constrained networks, cannot use the current solution provided by Truphone, detailed in Section 3.3, due to network overhead. The proposed solution is based on the current working solution provided by Truphone and it is designed specifically for constrained devices.

4.2 Building Blocks

One of the key blocks of the proposed solution is **NB-IoT**. As seen in Section 2.7, NB-IoT was purposely designed for constrained devices and environments. This network is here considered to connect the devices to the Internet.
The second building block is **CoAP**. CoAP was introduced in Section 2.8 and is built specifically to cope with constrained devices and environments. This solution uses non-confirmable messages as the acknowledgement messages introduced by the reliability mechanism would originate unnecessary overhead to the protocol. The lightweight nature of this protocol, when compared

to other application-layer protocols, makes it the best alternative to use. It is used when the devices need to send or receive data from other entities.
The third and final building block is **CBOR** and, more specifically, **CWT**, detailed in Section 2.11 and 2.11, respectively. The use of CWTs will allow the creation of a more compact access token.

4.3 Intermediate Solution 1

The first intermediate solution is an adaptation of the existing solution. It uses the same protocol participants as the "notarizer". In order to reduce traffic network on the client side this first approach uses different data transfer protocols and a different method to encode the token. This solution presents four major changes:

1. It uses CoAP instead of using HTTP when the client requests the token from the authorization server
2. It uses a CWT instead of a JWT to encode the access token
3. Token validation is now performed by the authorization server
4. The client no longer sends SIM data to the authorization server. The authorization server can associate an IP to SIM data so the client's IP is sufficient for the authorization server to identify a client, since they share Truphone's private network.

The first two changes will reduce the client's traffic network. By using a lighter application-layer protocol and a more efficient data format for the access token, the client will transmit fewer bytes when it requests the token, when it receives it and when it publishes the message in the third-party as the token is present in that message. The third change tackles the need to manually distribute the authorization server's public key as the token validity is now confirmed by the authorization server. The access token is equal to the one used in the current solution apart from the different data formats. The last change will allow the token request to be slightly smaller since there is no need to send any payload.

4.3.1 Protocol Flow

The protocol flow is represented in Figure 4.1. In **(1)** and **(2)**, the client requests the token to the authorization server. These two messages are similar to the ones made in the "notarizer" flow with the already mentioned changes of using CoAP instead of HTTP and the use of a CWT instead of a JWT to encode the token. Message **(3)** is the same as in the "notarizer" sending

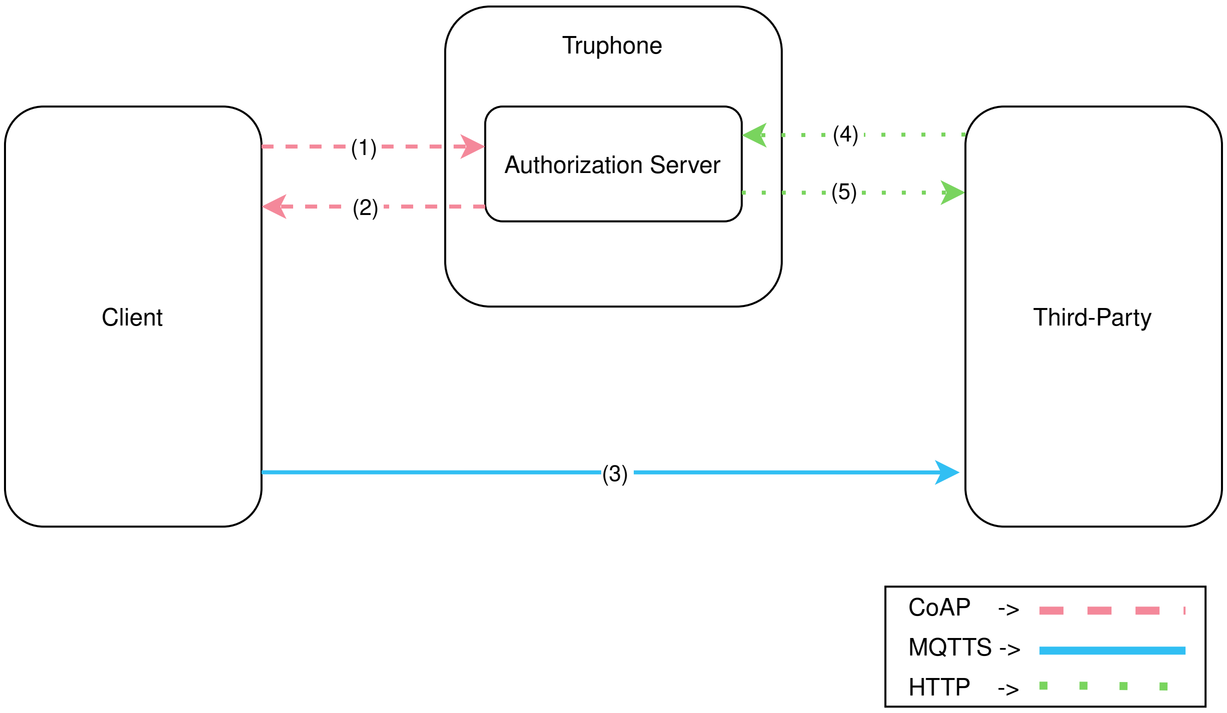

Figure 4.1: Intermediate Solution 1 Flow

both the token and the message to publish to the third-party using MQTTS. Messages **(4)** and **(5)** are introduced here. In **(4)** the third-party sends the access token received in **(3)** to the authorization server and the authorization server will assess the validity of the token and send it back to the third-party in **(5)**. This flow is triggered when the client needs to publish a message to a third-party.

4.3.2 Limitations and Advantages

While this intermediate solution solves some of the "notarizer" limitations, like the need to manually distribute the authorization server's public key and the use of HTTP, it still has a problem: the use of MQTTS is still an issue that brings unacceptable communication overhead for a constrained device to use, since it runs on TCP and it uses TLS to secure communications.

4.3.3 Security

To prove the security of intermediate solution 1, the changes it introduces from Truphone's existing solution will be analyzed. Truphone's existing solution is proved secure in Section 3.3.5. Intermediate solution 1 implements the following changes compared to the "notarizer":

1. It uses CoAP instead of using HTTP when requesting the token

2. It uses a CWT instead of a JWT to encode the access token

3. Token validation is now performed by the authorization server

4. The client no longer sends SIM data to the authorization server. The authorization server can associate an IP to SIM data so the client's IP is sufficient for the authorization server to identify a client, since they share Truphone's private network.

The first change is inconsequential as the security comes from the private network, not from the application-layer protocol chosen here. The second change also does not affect security, since the CWT is properly signed as it was before. The third change is not a problem as token validation can be done by any party that possesses the public key from the key pair used to sign the access token. As long as the connection between the third-party and the authorization server is secure and the third-party trusts the authorization server, this is secure. The fourth change is meaningless to security, and as long as the client can send some information that the authorization server can use to identify and authenticate the client, this is not a problem.

4.4 Intermediate Solution 2

To improve the above solution, in particular the use of MQTTS. This intermediate solution introduces a new protocol participant, the broker. The broker connects the client to the third-party service, as illustrated in Figure 4.2. It is introduced to allow the client to use CoAP even if the third-party is not prepared to use it. It acts as a middle-man between the client and the third-party service, receiving CoAP requests from the client and extracting the information meant to the third-party service crafting, from this, an appropriate request is sent to the third-party. The type of request depends on the third-party. In the case of the current solution, the third-party, as mentioned before, uses MQTTS. In this case, the broker receives a CoAP request, extracts its information and uses MQTTS to send this information the third-party on behalf of the constrained device. The addition of the broker is a crucial step as it enables the use of constrained devices operating in constrained environments by allowing the device to communicate exclusively using CoAP, which, as seen before, is the best fit for this kind of device. The broker is also deployed in Truphone's private network and so, it has access to the same features as the authorization server mentioned in Section 3.3.1, namely a private confidential connection to the client. The access token and its validation process is the same to the one used in intermediate solution 1.

4.4.1 Protocol Flow

The protocol flow, depicted in Figure 4.2. Messages **(1)** and **(2)** stay the same as in the first intermediate solution. Message **(3)**, as in the first intermediate solution, sends the access token and the message, however, in this case, the client does not send this directly to the third-party. It sends it to the broker which will then send this information to the third-party on behalf of the client. It also uses CoAP instead of MQTTS. Since MQTTS has a reliability mechanism and CoAP does not, the broker sends the message **(4)** to confirm that it received the message, and when paired with a timeout it can be used to create an application-layer reliability mechanism. In message **(5)** the broker sends the access token and the message to the third-party using MQTTS as the third-party dictates. Messages **(6)** and **(7)** perform token validation. The third-party sends the access token to the authorization server, that responds with the assessment of the token's validity. This token validation process is the same as in the first intermediate solution and stay the same. This flow is triggered when the client needs to publish a message to a third-party.

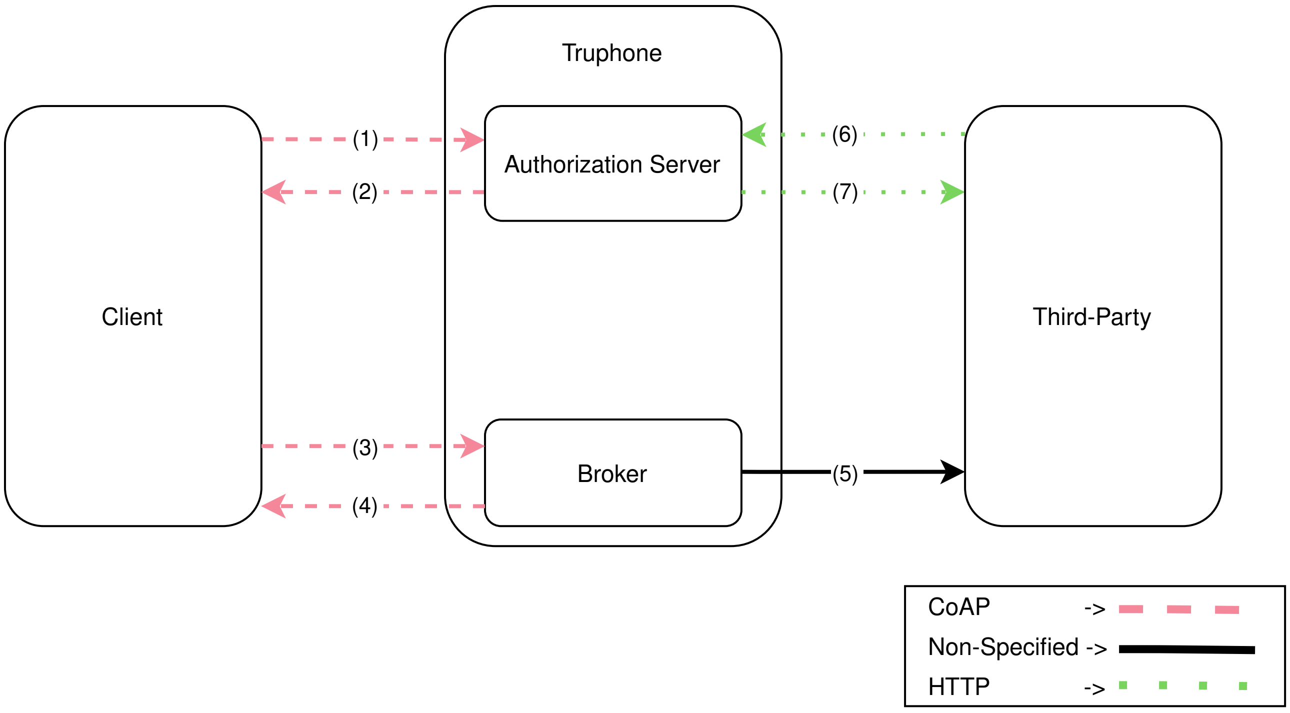

Figure 4.2: Intermediate Solution 2 Flow

4.4.2 Limitations and Advantages

This solution does not have any major limitations like the ones presented before. The use of CoAP instead of MQTTS requires the aforementioned application layer reliability mechanism to be implemented. The broker introduces a layer of abstraction that allows a client to send a message to a third-party in a transparent way. One use case where the broker can be useful is in managing the client. Since the client is expected to be a constrained device, it is reasonable to assume that there will be some entity managing it. In the case of migrating a client or a collection of clients from a third-party to another (for example from AWS to another cloud provider), this can be done without changes in the client since the broker will deliver these messages, and just needs to be informed of the change so it can behave accordingly.

4.4.3 Security

To prove the security of intermediate solution 2, the changes it introduces from intermediate solution 1 will be analyzed. Intermediate solution 1 is proved secure in Section 4.3.3.
Intermediate solution 2 implements the following changes compared to intermediate solution 2:

1. The introduction of the broker

2. The use of CoAP instead of MQTTS by the client

Starting with the first change. The broker is deployed inside Truphone's private network and so its connection to the client benefits from the same guarantees as the connection between the client and the authorization server. The connection between the broker and the third-party is made via MQTTS, which, as seen before, is secure.

4.5 Final Proposed Solution

The proposed solution optimizes the second intermediate solution by delegating the responsibility of requesting a token from the client to the broker. This approach allow for a reduction of the traffic network on the client side since messages **(1)** and **(2)** from the previous solution will not be present in this solution, as illustrated in Figure 4.3. In this approach the token request is made by the broker.

4.5.1 Protocol Flow

The protocol flow is represented in Figure 4.3. In message **(1)**, the client sends to the broker the data it needs to publish in the third-party. The broker confirms the reception of this message with **(2)** for reliability. The broker will then extract the client's IP and send it in **(3)** a token request to the authorization server which the authorization server can identify, as mentioned before. In **(4)**, the authorization server sends the access token identifying the client to the broker which will then compose a message using this token and the message sent by the client in **(1)** to the third-party, on behalf of the client, in **(5)**. Messages **(6)** and **(7)**, where the third-party requests the authorization server to assess the validity of the access token, will stay the same, as in the previous solutions.

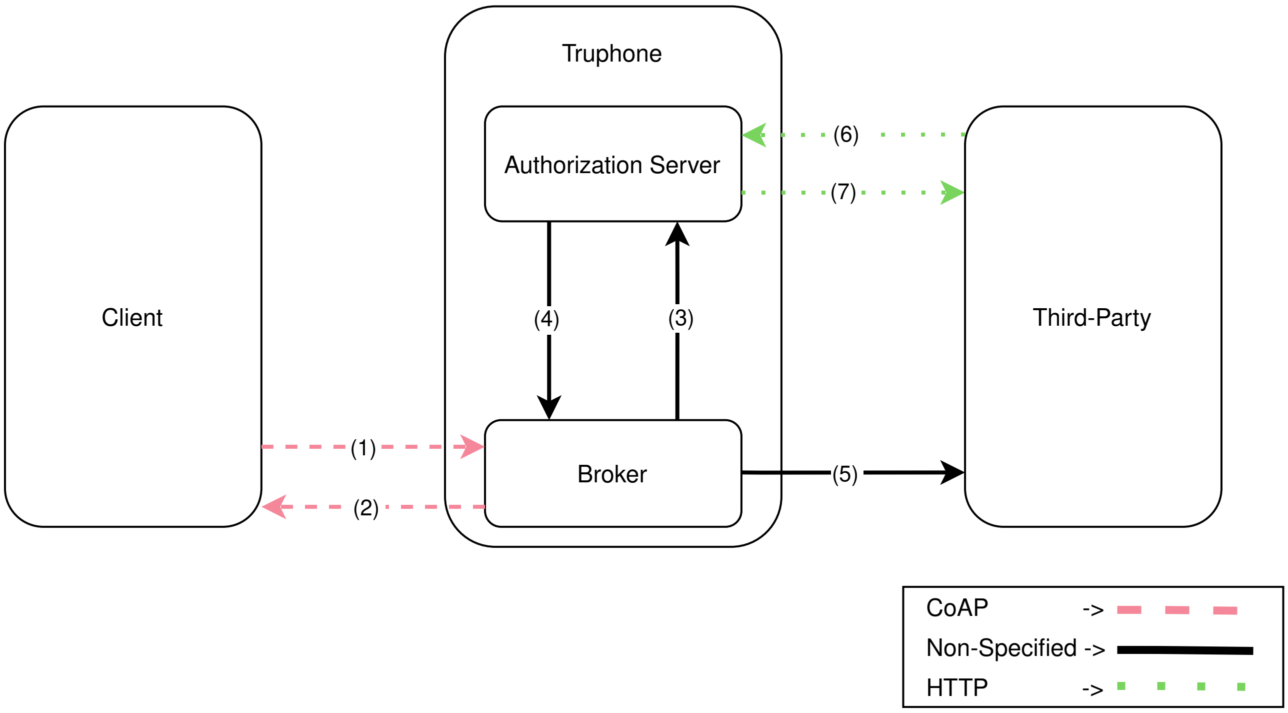

Figure 4.3: Proposed Protocol Flow

4.5.2 Limitations and Advantages

By delegating the token request to the broker, this protocol essentially builds a transparent authentication scheme for the client. The client only needs to send to the broker the message that it wants to publish to the third-party and the broker will deal with everything from here. By doing this, the solution optimizes traffic network on the client-side as it only needs to send the message that it would need to send anyway and it can do it in a near-optimal way by using CoAP instead of other heavier protocols. It also allows for client managing like in intermediate solution 2.

4.5.3 Security

To prove the security of the final proposed solution, the changes it introduces from intermediate solution 2 will be analyzed. Intermediate solution 2 is proved secure in Section 4.4.3.
The final proposed solution implements the following changes compared to intermediate solution 2:

1. The delegation of the token request from the client to the broker

This change does not affect security since the connection between the authorization server and the broker is made on Truphone's private network. The broker uses the client's IP to identify it to the authorization server like before and so there is no change to the protocol's security, proving this solution to be secure.

4.6 Access token

The access token, as mentioned in Section 4.2, is a CWT. As depicted in Section 2.11, a CWT must be either signed, MACed or encrypted. The access token must preserve its integrity and it must be verifiable that it was issued by the authorization Server. Given these requirements, the access token will be signed by the authorization Server. This presents another choice that must be made: should it be signed using an ECDSA or an EdDSA algorithm. According to [43], the curve P-256 should be used to generate keys used for digital signatures for authentication purposes. A P-256 curve generates a 64-byte key. As per [44], ECDSA signatures are 2 times longer than the signer's private key for the curve used during the signing process. This means that using ECDSA would generate a 128 byte sized signature. According to [45], EdDSA can use two curves for key generation: the edwards25519 and edwards448 curves. EdDSA public keys have exactly b bits and EdDSA signatures have exactly 2b bits. The value b is a multiple of 8, therefore, the public key and signature lengths are an integral number of octets. For Ed25519, b is 256, so the private key is 32 octets. With a private key of 32 bytes, EdDSA using the edwards25519 curve would generate a signature with 64 bytes. Since both algorithms are secure [45], the access token is signed using EdDSA with the edwards25519 curve, also known as an Ed25519 digital signature, to produce a smaller signature.
Apart from the signature, the access token contains claims about the subject that possess it. As seen in Sections 2.10 and 2.11, unlike a JWT, a CWT does not have any mandatory claims that must be made about a subject. The access token provided by the current solution contains the

claims mandatory as per the OAuth 2.0 standard. This includes the issuer, subject, expiration date and audience claims. The access token in the proposed solution contains the same claims. The claim “subject” needs to uniquely identify a subject so the authorization server uses the ICC-ID and the IMSI of the client in this claim in the following way: “ICC-ID”—“IMSI”. This value will allow any client to be uniquely identified.

Validating an Access Token

The tokens are validated by the authorization server. The validation process of an access token has 4 steps:

1. It is necessary to validate the Ed25519 digital signature, assuring that it was signed with a Truphone’s private key
2. It is necessary to confirm that the issuer claims match “Truphone”
3. It is necessary to confirm if the access token is not expired
4. It necessary to confirm if the audience claim matches the identity of who wants to validate the token

If any of these steps fail, the token is considered invalid.

4.7 Improvements

The proposed solution was designed to solve the current’s solution shortcomings identified in Section 3.3, namely:

1. The use of HTTP
2. The use of JWT
3. The use of MQTTS
4. Lack of third-party interoperability

To solve issue 1., the proposed solution uses CoAP. To solve issue 2., the proposed solution uses CWT encoded tokens. Issue 3. is directly related to issue 4. as it originates from a third-party restriction. To deal with issues 3. and 4., the broker is introduced in the proposed solution. It allows the client to use CoAP for every communication and it eliminates the hassle of modifying multiple clients to deal with different third-parties. It delegates this effort to the broker making this process easier as one broker will serve multiple devices.

4.8 Implementation Details

4.8.1 Third-party service

The third-party used in all implementations (including the baseline) is AWS IoT Core. AWS IoT Core is a cloud service from AWS made for IoT devices. The use of this third-party shapes this implementation as AWS IoT Core requires connecting devices to use either HTTP, MQTT or LoRaWAN. Keeping in mind the objective to minimize traffic network, the connection to AWS IoT Core was made in MQTT. As explored in Section 2.9, MQTT was made specifically for constrained devices and it is lighter than HTTP. LoRaWAN is a low power wide area networking protocol made for IoT connectivity [46], however, it requires additional infrastructure, namely LoRaWAN gateways and network servers. This brings additional constraints to the client which would need to set up this infrastructure, making this option a less flexible alternative than using CoAP. AWS IoT Core requires MQTTS to be used. This means that the connection with AWS is protected by the use of TLS version 1.2 and AWS is authenticated to the client, the messages exchanged are encrypted and their integrity is protected. In Section 2.9, MQTT and MQTTS are exposed with more detail.

4.8.2 Authorization Server

The authorization server runs on Truphone's network and it has access to a mapping between devices' IP addresses and ICC-IDs. The authorization server is built in Rust. Rust is a modern, statically typed language that offers better memory usage than other modern and statically typed languages (Java and Go) [47]. Due to business constraints, excessive memory usage is a concern (higher cost) and Rust also supports CoAP and CBOR (by using the coap [48] and serde_cbor [49] crates). Rust however lacks support for CWT usage which dictated the need to develop a crate that added this feature. The cwt crate was developed on top of the aforementioned serde_cbor crate which offers serialization and deserialization support for CBOR objects, however, it can not create or read COSE headers that, as described in Section 2.11, are necessary for CWTs.

4.8.3 Broker

The broker is a service built in Rust deployed on Truphone's network. It receives CoAP requests from the clients and redirects them to the third-party service using MQTTS. In this proof-of-concept, only one third-party will be used and so the broker contains the needed CA certificate to verify the third-party identity. In future use, the broker should be agnostic to the third-party.

Chapter 5

Evaluation

This chapter presents the evaluation of the proposed solutions and the obtained experimental results. It starts with the introduction of the considered metrics, and then details how the tests were set up and presents their results. The chapter concludes with an evaluation of the experimental results and a discussion on the effectiveness of each solution.

5.1 Metrics

To test the effectiveness of the proposed solution and to validate the improvements estimated in Section 4.7 a network traffic test is considered.
Constrained devices in constrained environments need to minimize the amount of data exchanged. By measuring the traffic in the network, it is possible to test the effectiveness of the proposed solution.
Network traffic is affected by the amount of data in the network. This data can be separated into two distinct parts: the payload and the network overhead. The payload is the content of a message. The network overhead is all the headers of lower-layer protocol data and information that is necessary for the proper transmission of a message. In the case of the proposed solution, the payload size is related to the token size and message size. By measuring token size, it is possible to understand both the improvements that this solution provides, when compared with other authentication mechanisms, and the overhead impact on the network's traffic and how the use of CoAP affects the message size when compared to other transport-layer protocols. It is also interesting to see the differences in data transmission at each OSI layer and for the data received and sent by the client, since both operations have different costs for IoT devices. Another interesting discussion is to assess if the proposed solutions are suited for NB-IoT.

5.2 Test set-up

To test the network traffic the following set-up was used for every tested solution ("notarizer", intermediate solutions 1 and 2 and the final solution, described in Chapter 4). The client was run on a PC connected to the Internet. Wireshark was used to collect every packet sent between the client and the authorization server and between the client and the third-party (or broker in the case of the second prototype). Every test was performed 10 times. The network's conditions, depicted in Table 5.1, were also tested, which depict the packet loss and RTT (Round-Trip Time). This was obtained by pinging the authorization server, the broker and the third-party 100 times.

	Client to Authorization Server	Client to Third-Party	Client to Broker
Average RTT (ms)	55	74	62
Packet Loss (%)	0	0	0

Table 5.1: Network Conditions

By using Wireshark in the client, it is possible to test the entire traffic network on the client-side which is what is important since the client is the constrained node that needs to minimize the network traffic. It also allows the examination of the traffic per layer and to examine the data contents, allowing for an analysis of not only the overhead in each layer but also the difference in data (excluding protocol overhead) sent in each solution which corresponds to the token size and message sent. In every test, the message published by the client was a 3-byte message.

5.3 Experimental Results

The experimental results are presented per tested solution and are divided into two flows, in order to better assess the impact of each particular change. The first flow is the "Get Token" flow in which the client contacts the authorization server to request the token. The second flow is the "Client Publish" flow where the client sends the message, either to the broker or directly to the third-party. They are also divided by sent and received data by the client-side.
Starting by presenting the average bytes, with the standard deviation, sent and received by the client in each layer for each flow for each iteration. There are two flows considered: the flow where the client requests the access token and the flow where the client sends a message to the third-party. These flows will be named as "Get Token" flow and "Publish" flow, respectively. Then a comparison between the total bytes received and sent in each iteration, will be presented.

5.3.1 “Get Token” Flow

Starting with the depiction of the “Get Token” flow. Table 5.2 depicts the bytes received and sent by the client. The same information, divided by OSI layer, is depicted in Figures 5.1 and 5.2

	Baseline	Solution 1	Solution 2	Solution 3
Received (B)	969 ± 0	179 ± 0	179 ± 0	0
Sent (B)	525 ± 0	55 ± 0	55 ± 0	0

Table 5.2: Bytes received and sent by the client in the “Get Token” flow for every solution

The decrease in traffic network is clear with the changes introduced in the first intermediate solution. By making simple modifications such as changing the application-layer protocol, the data format used for the token and by authenticating clients via IP instead of SIM data, an 81.53% reduction in bytes received and an 89.52% reduction in bytes sent was observed at intermediate solution 1 (and 2, that uses the same mechanism). Solution 3 is optimal since the client does not request the token leaving this task for the broker and so sending and receiving 0 bytes.

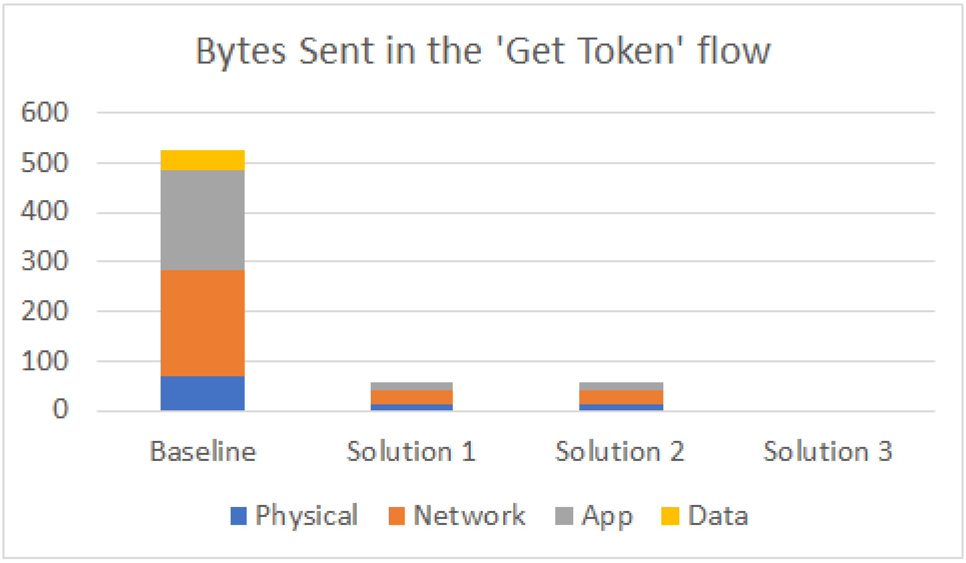

Figure 5.1: Bytes Sent in the “Get Token” flow

In Figure 5.1 it is possible to see a significant decrease, when comparing the Baseline to the proposed solutions, in bytes sent by the client when requesting the token. This decrease is explained by the use of CoAP instead of HTTP. In the physical layer, we see a decrease in bytes sent that is related to the decrease in the number of packets sent. In the network layer, the use of UDP instead of TCP, explains this decrease. In the application layer, the decrease is caused by the reduced overhead that CoAP has when compared to HTTP. In both prototypes the CoAP request does not carry any payload as the only information needed to identify a client is their IP in Truphone's private network. In the baseline, the client sends data to help with this identification, sending the SIM's ICC-ID and IMSI, which also confirm the client's identity. Figure 5.2 depicts the bytes received by the client in the "Get Token" flow. As in Figure 5.1, it is possible to see a decrease in bytes. Again, the decrease of bytes received in the physical layers is explained by the reduced number of packets received. In the network layer, it is explained by the use of UDP instead of TCP and, in the application layer, it is explained by the use of CoAP instead of HTTP. The reduction in data sent is explained by the usage of a CWT instead of a JWT to codify the access token.

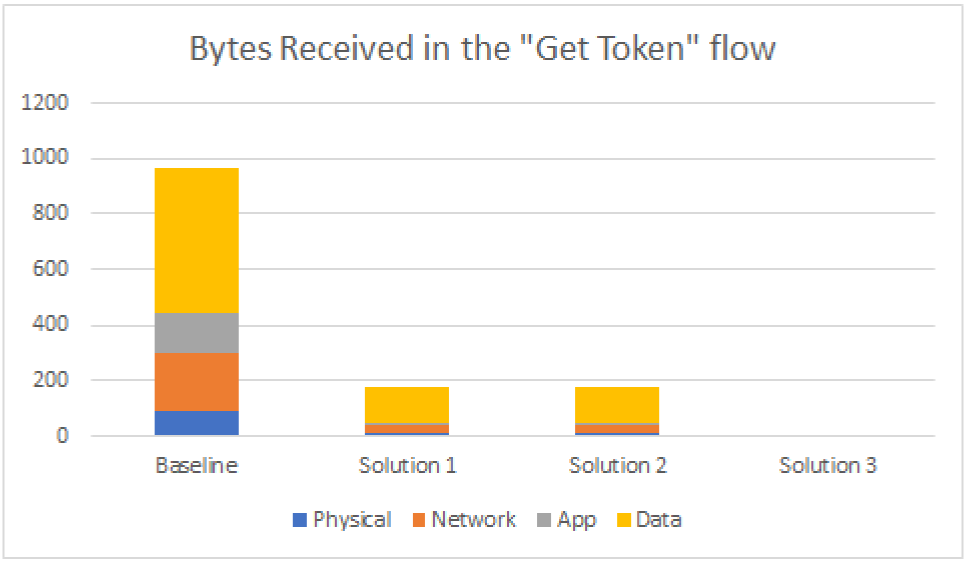

Figure 5.2: Bytes Received in the "Get Token" flow

5.3.2 "Client Publish" Flow

Now the depiction of the "Client Publish" flow. Table 5.3 shows the bytes received and sent by the client, and further detailed per OSI layer in Figures 5.3 and 5.4

	Baseline	Solution 1	Solution 2	Solution 3
Received (B)	6204.6 ± 37.757	6169.4 ± 46.419	52 ± 0	52 ± 0
Sent (B)	1859 ± 25.456	1512 ± 25.456	228 ± 0	55 ± 0

Table 5.3: Bytes received and sent by the client in the "Client Publish" flow for every solution

In this case, the decrease in traffic network is not so meaningful with the changes implemented in the first intermediate solution. Most changes were targeted at the "Get Token" flow, however, there is an 18.67% decrease in bytes sent by the client due to the changes to the token format, meaning that the token is smaller and so the client sends fewer data. There is a meaningless reduction of 0.57% that is statistically irrelevant. The significant changes come with the second intermediate solution that eliminates the costly use of MQTTS with the introduction of the broker. With a whopping 99.16% reduction in received bytes received and an 87.74% reduction in sent bytes, when compared to the baseline, showing that the intermediate solution 2 presents clear benefits to the client. The proposed solution eliminates the need to send the token to the broker which helps to enhance this gain from 87.74%, when compared to the baseline, to 97.04%.

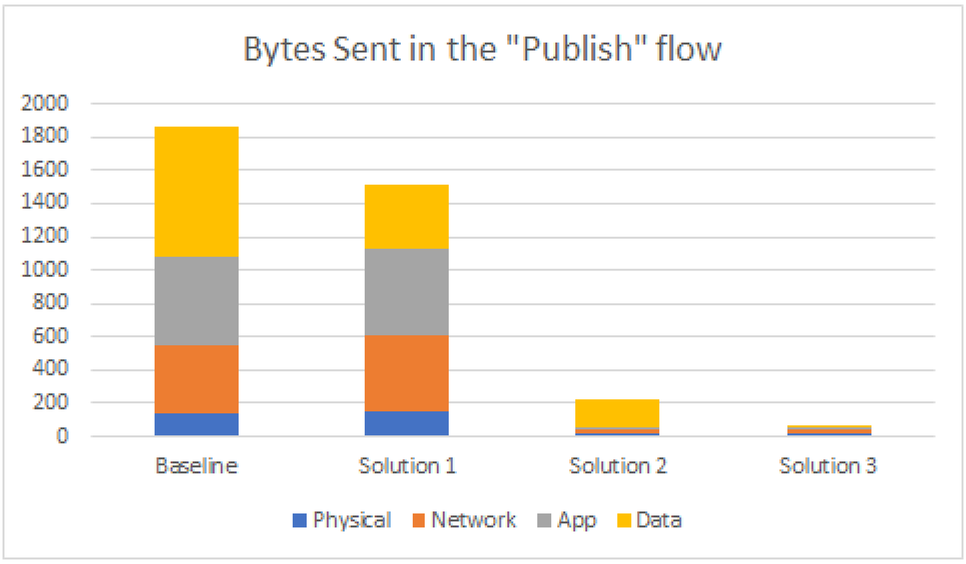

Figure 5.3: Bytes Sent in the “Client Publish” flow

In Figure 5.3 there is more nuance as there is a difference in performance, not only between the baseline and the intermediate solutions but also between the intermediate solutions themselves. The major difference between the baseline and the first intermediate solution is in the data which comes from the difference in technology used to codify the token. The use of CWT instead of JWT is responsible for this difference. The big changes are seen when comparing the first to the second intermediate solution, given the introduction of the broker. As discussed in Section 4.8.3, the introduction of the broker allows for the use of CoAP instead of using MQTTS. The use of MQTTS, which requires various handshakes and a heavy overhead, is replaced by a single CoAP request and this is apparent in the performance of the two prototypes. The proposed solution optimizes this flow since there is no need to send the token in the request.
In Figure 5.4 the difference in performance is explained by the fact that both the baseline and first intermediate solution use MQTTS and have to perform costly TCP and TLS handshakes where as the intermediate solution 2 and the proposed solution use CoAP. The big amount of data, when compared to other layers, is due to the need to receive the TLS certificate of the third-party.

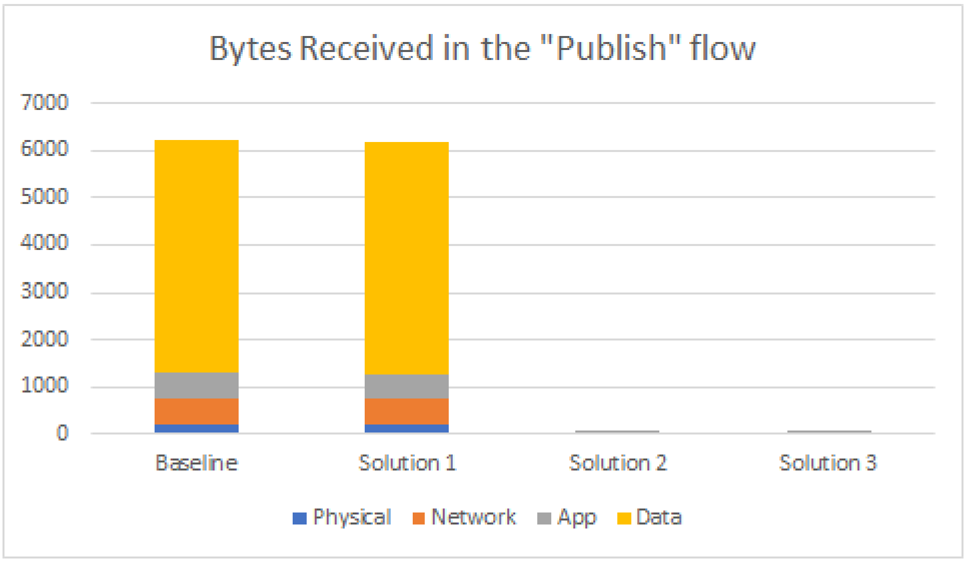

Figure 5.4: Bytes Received in the “Client Publish” flow

5.4 Evaluation

The results presented show that the proposed solution brings a massive increase in performance, with the intermediate solution 2 and the final proposed solution showing that the introduction of the broker comes with tremendous benefit for the device, as it allows the device to use CoAP for every communication effectively bypassing the third-parties restrictions. Another important analysis is to check if these prototypes are ready to use NB-IoT. As said in Section 2.7, an NB-IoT device is expected to transmit up to 200 bytes per day. In table 5.4 the total network traffic of the client is depicted.

	Baseline	Solution 1	Solution 2	Solution 3
Total Traffic Network (B)	9546.8 ± 36.199	7912 ± 52.679	517 ± 0	107 ± 0

Table 5.4: Total Traffic Network in the client for every solution

Assuming that a client needs to send a message to the third-party at least one time per day the only solution that is ready to use NB-IoT is the final proposed solution. The message sent in the tests was a 3-byte message which means that the total overhead for communications is 104 bytes, allowing the client to send one message per day to the third-party up to 96 bytes.

5.5 Overview

In this chapter, the evaluation of both Truphone’s existing solution and the proposed solutions based on the client-side traffic network is presented.

Truphone’s existing solution acted as a baseline, and with every proposed solution it was possible to see a decrease in client-side traffic network, with the most significant decrease being found when introducing the broker. The final proposed solution presented a decrease of more than 98% in total client-side traffic network when compared to Truphone's existing solution. The final proposed solution was also found to be fit for NB-IoT use.

Through the assessment of the final proposed solution it is possible to conclude that the this work fulfills every aforementioned requirement and it achieves the initially proposed goals.

Chapter 6

Conclusion

This chapter concludes this work by stating its achievements and some ideas for future work that can be interesting to optimize the solution presented.

6.1 Achievements

In this work aims to solve the problem of a lack of an automatic authentication mechanism for constrained IoT devices operating in constrained environments that allows devices to authenticate themselves towards third-party services is tackled. IoT devices are a emerging market characterized by the heterogeneity of its technology and the limitations of IoT devices. Traditional authentication mechanisms are not suited to deal with the limitations of some IoT devices. To address this problem this work proposes an adaptation to a OAuth 2.0 based Truphone's existing solution, that is capable of identifying devices equipped with an e-SIM by relying on their cellular authentication. The proposed final solution's main innovation is the introduction of a new key role in the protocol: the broker. The broker is deployed on Truphone's core network, and it shares a private connection with the client device. It can also identify a device by associating their IP address to their SIM credentials, that uniquely identify a device. The broker also manages the device's connection to the third-party, making it possible for the device to delegate the entirety of the authentication mechanism to the broker, who acts on the client's behalf. The final proposed solution also uses a more efficient data-format and it runs on protocols that present lower network overhead.
In this work, a new method to authenticate constrained devices operating in constrained environments to a third-party is presented. It relies on cellular authentication provided by Truphone and on the introduction of a broker to which the client delegates all authentication mechanisms. The proposed solution fulfils every requirement listed in the first chapter. It authenticates a

client based on his cellular connection, it reduces traffic network in the client by more than 50% and it allows for a seamless authentication mechanism, as the client only needs to send a message to the broker and the broker will deal with the authentication to the third-party.

6.2 Future Work

One idea for the improvement of the work developed in this thesis is to use CoAP on top of NIDD protocols. The use of NIDD instead of UDP will allow for the reduction of the traffic network on the client-side. The problem with its implementation is that currently authentication is done via the client's IP address. NIDD do not use IP addresses and so an alternative would have to be developed to deal with this constraint. An energy-consumption study of the client would also be interesting, especially when cross-referencing those results with the reductions in bytes sent and received, and how they affect a device's battery life. A response time analysis would also be valudable, to understand the impact of these changes on response times.

Bibliography

[1] S Jaiganesh, K Gunaseelan, and V Ellappan. Iot agriculture to improve food and farming technology. In *2017 Conference on Emerging Devices and Smart Systems (ICEDSS)*, pages 260–266. IEEE, 2017.

[2] L Siri Chandana and AJ Raja Sekhar. Weather monitoring using wireless sensor networks based on iot. *Int. J. Sci. Res. Sci. Technol*, 4:525–531, 2018.

[3] Mengru Tu. An exploratory study of internet of things (iot) adoption intention in logistics and supply chain management. *The International Journal of Logistics Management*, 2018.

[4] Wolfgang Rankl and Wolfgang Effing. *Smart Card Handbook*. Wiley Publishing, 4th edition, 2010. ISBN 0470743670.

[5] CSMG. Reprogrammable SIMs: Technology, Evolution and Implications. Technical report, Ofcom, 2012.

[6] GSMA. eSIM Whitepaper: The what and how of remote sim provisioning. Technical report, GSMA, March 2018.

[7] Michael Moorfield and Ruud Derwig. Securing the mobile iot. Technical report, Truphone, Synopsys Inc., May 2019.

[8] In Lee and Kyoochun Lee. The internet of things (iot): Applications, investments, and challenges for enterprises. *Business Horizons*, 58(4):431–440, 2015.

[9] Heiner Lasi, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. Industry 4.0. *Business & information systems engineering*, 6(4):239–242, 2014.

[10] The iot in 2030: Which applications account for the biggest chunk of the $1.5 trillion opportunity? `http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm`. Accessed: 2020-12-28.

[11] Muhamed Umar Farooq, Muhammad Waseem, Sadia Mazhar, Anjum Khairi, and Talha Kamal. A review on internet of things (iot). *International journal of computer applications*, 113(1):1–7, 2015.

[12] Neil Briscoe. Understanding the osi 7-layer model. *PC Network Advisor*, 120(2):13–15, 2000.

[13] Sumit Kumar, Sumit Dalal, and Vivek Dixit. The osi model: Overview on the seven layers of computer networks. *International Journal of Computer Science and Information Technology Research*, 2(3):461–466, 2014.

[14] DongJin Lee, Brian E Carpenter, and Nevil Brownlee. Observations of udp to tcp ratio and port numbers. In *2010 Fifth International Conference on Internet Monitoring and Protection*, pages 99–104. IEEE, 2010.

[15] Santosh Kumar and Sonam Rai. Survey on transport layer protocols: Tcp & udp. *International Journal of Computer Applications*, 46(7):20–25, 2012.

[16] Behrouz A Forouzan. *TCP/IP protocol suite*. McGraw-Hill Higher Education, 2002.

[17] Hugo Krawczyk, Kenneth G Paterson, and Hoeteck Wee. On the security of the tls protocol: A systematic analysis. In *Annual Cryptology Conference*, pages 429–448. Springer, 2013.

[18] S. Turner and T. Polk. Prohibiting secure sockets layer (ssl) version 2.0. RFC 6176, RFC Editor, March 2011. URL `http://www.rfc-editor.org/rfc/rfc6176.txt`. `http://www.rfc-editor.org/rfc/rfc6176.txt`.

[19] Sean Turner. Transport layer security. *IEEE Internet Computing*, 18(6):60–63, 2014.

[20] E. Rescorla. The transport layer security (tls) protocol version 1.3. RFC 8446, RFC Editor, August 2018. URL `https://www.rfc-editor.org/rfc/rfc8446.txt`. `https://www.rfc-editor.org/rfc/rfc8446.txt`.

[21] Nagendra Modadugu and Eric Rescorla. The design and implementation of datagram tls.

[22] Rapeepat Ratasuk, Nitin Mangalvedhe, Yanji Zhang, Michel Robert, and Jussi-Pekka Koskinen. Overview of narrowband iot in lte rel-13. In *2016 IEEE conference on standards for communications and networking (CSCN)*, pages 1–7. IEEE, 2016.

[23] 3GPP. Cellular system support for ultra-low complexity and low throughput internet of things (ciot). TR 45.820, 3GPP, 2015. November.

[24] Narrowband iot solution developer protocols guide. Technical report, T-Mobile.

[25] Z. Shelby, K. Hartke, and C. Bormann. The constrained application protocol (coap). RFC 7252, RFC Editor, June 2014. URL `http://www.rfc-editor.org/rfc/rfc7252.txt`. `http://www.rfc-editor.org/rfc/rfc7252.txt`.

[26] David Sembroiz, Sergio Ricciardi, and Davide Careglio. A novel cloud-based iot architecture for smart building automation. In *Security and Resilience in Intelligent data-Centric Systems and communication networks*, pages 215–233. Elsevier, 2018.

[27] Sergey Slovetskiy, Poornima Magadevan, Yun Zhang, and Sandeep Akhouri. White paper lightweight m2m 1.1: Managing non-ip devices in cellular iot networks, October 2018.

[28] Dipa Soni and Ashwin Makwana. A survey on mqtt: a protocol of internet of things (iot). In *International Conference On Telecommunication, Power Analysis And Computing Techniques (ICTPACT-2017)*, volume 20, 2017.

[29] Dinesh Thangavel, Xiaoping Ma, Alvin Valera, Hwee-Xian Tan, and Colin Keng-Yan Tan. Performance evaluation of mqtt and coap via a common middleware. In *2014 IEEE ninth international conference on intelligent sensors, sensor networks and information processing (ISSNIP)*, pages 1–6. IEEE, 2014.

[30] Andy Stanford-Clark and Hong Linh Truong. Mqtt for sensor networks (mqtt-sn) protocol specification. *International business machines (IBM) Corporation version*, 1(2), 2013.

[31] T. Bray. The javascript object notation (json) data interchange format. RFC 8259, RFC Editor, December 2017. URL `https://www.rfc-editor.org/rfc/rfc8259.txt`. `https://www.rfc-editor.org/rfc/rfc8259.txt`.

[32] Javier Luis Cánovas Izquierdo and Jordi Cabot. Discovering implicit schemas in json data. In *International Conference on Web Engineering*, pages 68–83. Springer, 2013.

[33] Y. Sheffer, D. Hardt, and M. Jones. Json web token best current practices. RFC 8725, RFC Editor, December 2017. URL `https://www.rfc-editor.org/rfc/rfc8725.txt`. `https://www.rfc-editor.org/rfc/rfc8725.txt`.

[34] M. Jones, B. Campbell, and C. Mortimore. Json web token (jwt) profile for oauth 2.0 client authentication and authorization grants. RFC 7523, RFC Editor, May 2015. URL `http://www.rfc-editor.org/rfc/rfc7523.txt`. `http://www.rfc-editor.org/rfc/rfc7523.txt`.

[35] P. Hoffman and C. Bormann. Concise binary object representation (cbor). RFC 8949, RFC Editor, December 2020. URL https://www.rfc-editor.org/rfc/rfc8949.txt. https://www.rfc-editor.org/rfc/rfc8949.txt.

[36] Markel Iglesias-Urkia, Diego Casado-Mansilla, Simon Mayer, Josu Bilbao, and Aitor Urbieta. Integrating electrical substations within the iot using iec 61850, coap, and cbor. *IEEE Internet of Things Journal*, 6(5):7437–7449, 2019.

[37] M. Jones, E. Wahlstroem, S. Erdtman, and H. Tschofenig. Cbor web token (cwt). RFC 8392, RFC Editor, May 2018. URL https://www.rfc-editor.org/rfc/rfc8392.txt. https://www.rfc-editor.org/rfc/rfc8392.txt.

[38] J. Schaad. Cbor object signing and encryption (cose). RFC 8152, RFC Editor, July 2017. URL https://www.rfc-editor.org/rfc/rfc8152.txt. https://www.rfc-editor.org/rfc/rfc8152.txt.

[39] D. Hardt. The oauth 2.0 authorization framework. RFC 6749, RFC Editor, October 2012. URL http://www.rfc-editor.org/rfc/rfc6749.txt. http://www.rfc-editor.org/rfc/rfc6749.txt.

[40] Ludwig Seitz, Göran Selander, Erik Wahlstroem, Samuel Erdtman, and Hannes Tschofenig. Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth). Internet-Draft draft-ietf-ace-oauth-authz-42, Internet Engineering Task Force, June 2021. URL https://datatracker.ietf.org/doc/html/draft-ietf-ace-oauth-authz-42. Work in Progress.

[41] Urs Gerber. Authentication and authorization for constrained environments. Master's thesis, University of Zurich, 2018.

[42] G. Selander, J. Mattsson, and F. Palombini. Object security for constrained restful environments (oscore). RFC 8631, RFC Editor, July 2019. URL https://www.rfc-editor.org/rfc/rfc8631.txt. https://www.rfc-editor.org/rfc/rfc8631.txt.

[43] Elaine B. Barker and Quynh H. Dang. Recommendation for key management part 3: Application-specific key management guidance. Technical report, 2015. URL https://doi.org/10.6028/NIST.SP.800-57Pt3r1.

[44] Svetlin Nakov. *Practical Cryptography for Developers*. Gitbook, 2018. ISBN 978-619-00-0870-5. https://cryptobook.nakov.com/.

[45] Elaine Barker. Digital signature standard (dss). Technical report, Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards and Technology, Gaithersburg, MD, October 2019.

[46] Jonathan de Carvalho Silva, Joel JPC Rodrigues, Antonio M Alberti, Petar Solic, and Andre LL Aquino. Lorawan—a low power wan protocol for internet of things: A review and opportunities. In *2017 2nd International Multidisciplinary Conference on Computer and Energy Science (SpliTech)*, pages 1–6. IEEE, 2017.

[47] Dexter Darwich. Comparison between java, go, and rust. `https://medium.com/@dexterdarwich/comparison-between-java-go-and-rust-fdb21bd5fb7c`. Accessed: 2021-03-26.

[48] `https://github.com/Covertness`. coap. `https://crates.io/crates/coap`.

[49] `https://github.com/pyfisch`. serde_cbor. `https://crates.io/crates/serde_cbor`.