Pontifícia Universidade Católica
do Rio de Janeiro

Gabriel Brito Cantergiani

EdgeSec – A Security framework for middlewares and edge devices in the Internet of Things (IoT)

Dissertação de Mestrado

Dissertation presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor: Markus Endler

Co-Advisor: Anderson Oliveira da Silva

Rio de Janeiro

August 2023

PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Gabriel Brito Cantergiani

EdgeSec – A Security framework for middlewares and edge devices in the Internet of Things (IoT)

Dissertation presented to the Programa de Pós-Graduação em Informática, do Departamento de Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática. Approved by the Examination Committee:

Prof. Markus Endler
Advisor
Departamento de Informática – PUC-Rio

Prof. Anderson Oliveira da Silva
Co-Advisor
Departamento de Informática – PUC-Rio

Prof. Sérgio Colcher
Departamento de Informática – PUC-Rio

Prof. Alexandre Malheiros Meslin
Departamento de Informática – PUC-Rio

Prof. Hilder Vitor Lima Pereira
UNICAMP

Rio de Janeiro, August 25th, 2023

Gabriel Brito Cantergiani

Graduated in Computer Engineering by PUC-Rio (Rio de Janeiro, Brazil) in 2020.

Bibliographic data

Cantergiani, Gabriel Brito

EdgeSec : a security framework for middlewares and edge devices in the Internet of Things (IoT) / Gabriel Brito Cantergiani ; advisor: Markus Endler ; co-advisor: Anderson Silva. – 2023.
51 f. : il. color. ; 30 cm

Dissertação (mestrado)–Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2023.
Inclui bibliografia

1. Informática – Teses. 2. Segurança da informação. 3. Internet das coisas. 4. Criptografia. 5. Framework. 6. Edge computing. I. Endler, Markus. II. Silva, Anderson. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. IV. Título.

CDD: 004

To my family José, Glaucia, Julia, Ana, and Daniela, for all their support and love during these years.

Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

I would like to thank my advisors, Markus Endler and Anderson Silva for all the great discussions, suggestions, reviews, and overall help in achieving this goal.

I would also like to thank my friends and partners during the Masters Course, Matheus Leal and Mariana Salgueiro, for making this journey more enjoyable.

Abstract

Brito Cantergiani, Gabriel; Endler, Markus (Advisor); Silva, Anderson (Co-Advisor). **EdgeSec – A Security framework for middlewares and edge devices in the Internet of Things (IoT).** Rio de Janeiro 2023. 51p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

The importance of the Internet of Things (IoT) has increased significantly in recent years, and IoT devices are being used in many different industries and types of applications, such as smart homes, industrial sensors, autonomous vehicles, personal wearables, and more. While this brings technology innovation, new user experiences, and new business solutions, it also raises important concerns related to information security and privacy. In this work we present EdgeSec Framework, a new IoT security framework, made concrete as a security solution for ContextNet and Mobile-Hub middlewares. Its main goal is to extend and improve on an existing security architecture and implementation, creating a more generic, robust, and flexible solution that ensures authentication, authorization, data integrity and confidentiality. The framework was designed with full extensibility in mind by introducing protocol interfaces that can be implemented by external plugins, making it compatible to a variety of security algorithms and edge devices. A complete implementation was developed as proof-of-concept, and performance tests and experiments were made to evaluate the feasibility of the solution. Results show that EdgeSec framework can greatly improve the security of Mobile-Hub and similar IoT middlewares by increasing its compatibility and flexibility, and ensuring all the basic security protections.

Keywords

Security; Information Security; Internet of Things; Cryptography; Framework; Bluetooth; Edge Computing;

Resumo

Brito Cantergiani, Gabriel; Endler, Markus; Silva, Anderson. **EdgeSec – Um framework de Segurança para middlewares e dispositivos na Internet das Coisas.** Rio de Janeiro 2023. 51p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A importância da Internet das Coisas (IoT) tem aumentado significativamente nos últimos anos, e dispositivos IoT têm sido usados em diferentes indústrias e tipos de aplicação, como casas inteligentes, sensores indutriais, veículos autonomos, wearables, etc. Apesar deste cenário trazer inovações tecnológicas, novas experiências para usuários, e novas soluções de negócio, também levanta preocupações relevantes relacionadas a segurança da informação e privacidade. Neste trabalho nós apresentamos o EdgeSec Framework, um novo framework de segurança para IoT desenvolvido como uma solução de segurança para os middlewares ContextNet e Mobile-Hub. O seu objetivo principal é estender e melhorar uma arquitetura e uma implementação já existentes para estes middlewares, criando uma solução mais genérica, robusta e flexível,e garantindo autenticação, autorização, integridade e confidencialidade de dados. O framework foi elaborado com foco na total extensiblidade através da introdução de interfaces de protocolos, que podem ser implementadas por plugins, tornando-o compatível com uma variedade de algoritmos de segurança e dispositivos IoT. Uma implementação completa foi realizada como prova de conceito, e testes de desempenho e experimentos foram realizados para avaliar a viabilidade da solução. Os resultados mostram que o EdgeSec Framework pode melhorar significativamente a segurança do Mobile-Hub e diversos tipos de aplicações IoT através de uma maior compatibilidade e flexibilidade, e garantindo todas as proteções básicas de segurança.

Palavras-chave

Segurança; Segurança da Informação; Internet das Coisas; Criptografia; Framework; Bluetooth; Computação Edge;

Table of contents

List of figures

List of abbreviations

BLE – Bluetooth Low Energy

HMAC – Hash Message Authentication Code

ID – Identification

IDE – Integrated Development Environment

IDS – Intrusion Detection Systems

IoMT – Internet of Mobile Things

IoT – Internet of Things

JAR – Java Archive

MAC – Message Authentication Code

NFV – Network Function Virtualization

OTP – One Time Password

PAN – Personal Area Networks

POC – Proof-of-concept

RSF – Reconfigurable Security Function

SA – Security Agent

SDN – Software Defined Networks

SF – Security Function

TLS – Transport Layer Security

VPN – Virtual Private Network

WLAN – Wireless Local Area Networks

WPAN – Wireless Personal Area Networks

1

Introduction

The importance of the Internet of Things (IoT) has increased significantly in recent years, and IoT devices are being used in many different industries and types of applications, such as smart homes, industrial sensors, autonomous vehicles, personal wearables, and more. Further growth is expected for the coming years, with new sensors becoming cheaper, more powerful, and more reliable [1]. This opportunity may be used to drive technological advancements such as in 5G networks, Edge Computing and Artificial Intelligence. According to some projections, the impact of IoT on the global economy may reach $11 trillion by 2025, with more than 100 billion connected devices [2].

While IoT brings technology innovation, new user experiences, and new business solutions, it also raises important concerns related to information security and privacy. In today's connected and digital world, any device that exchanges data through the internet is exposed to many of the most common threats. This includes unauthorized access to confidential information, private data breaches and unauthorized control of devices to perform malicious activities on the internet.

However, because IoT smart devices allow for remote interaction with the physical world (e.g., smart locks, ambient temperature control, robots, vehicles, etc.), compromises in their security can pose other types of risks that involve the physical safety of people and infrastructure. Additionally, the large-scale and distributed nature of many IoT systems allows for attacks that are potentially very destructive and can have an enormous impact on internet services.

Traditional information security mechanisms that involve perimeter security and devices like firewalls and intrusion detection systems, were created without this new IoT distributed context in mind. The level of protection required by some IoT systems cannot be ensured using these same security mechanisms. That is why the massive adoption of IoT in society and industries brings new challenges and the need for new security solutions aimed at the Internet of Things.

The focus of this research is to define and present a new security framework for IoT middlewares, which we named EdgeSec Framework. Its development was made concrete as a solution for improving the security of ContextNet and Mobile-Hub middlewares [3] [4], and it can be considered an evolution of previous research on this area. It is built on top of an existing security architecture and algorithm proposed in the context of the Internet of Mobile Things and Edge Computing, using it as conceptual foundation [5]. It is also based on a previous implementation project called EdgeSec, which aimed at creating a practical and functional prototype of the same security architecture and incorporate it inside Mobile-Hub middleware [6].

The main goal of EdgeSec Framework is to extend and improve on the existing EdgeSec implementation, creating a more generic, flexible, and extensible solution. This should increase compatibility with different IoT devices and enforce basic security protections on IoT middlewares such as the Mobile-Hub. Additionally, it also has the goal of (i) refactoring and optimizing previous implementations of the security architecture; (ii) porting it to new versions of

Mobile-Hub middleware; and (iii) improving performance metrics to create a feasible, flexible, and practical security solution.

Because Mobile-Hub is a mobile middleware, it can be used for crowdsensing, by offering connectivity to a wide array of devices, each possessing varying hardware setups, manufacturers, and network providers. To support this kind of scenario, the Framework was developed with extensibility was a key attribute considered by design, allowing for new security protocols and algorithms to be easily added to the system. This was achieved through the definition of multiple protocol interfaces and implementation of plugins. Protocol suite negotiations were also introduced into the main algorithm, together with many other security mechanisms that ensure authentication, authorization, integrity, and confidentiality to all data exchanged between devices.

In this dissertation, we first go through some definitions and concepts to give background information on the proposed solution, which includes Internet of Mobile Things (IoMT), Edge Computing, Bluetooth Low Energy, essential security mechanisms, Mobile-Hub middleware and more. Then, in chapter 3 we present some of the related work on security frameworks for IoT. On chapter 4 we give a detailed explanation of how the security architecture and algorithm of EdgeSec works, and how the framework was structured to improve on existing solutions. Chapter 5 describes a prototype implementation created to prove the concept and run performance tests. These tests are presented and analyzed in chapter 6, and we give final remarks and conclusions on chapter 7.

2

Background and Definitions

2.1

IoMT and Edge Computing

The Internet of Things (IoT) describes physical objects with sensors, processing power, software and other technologies that connect and exchange data with other devices over communication networks. Because of their embedded technology, these devices are usually called *smart objects*, which is how we are going to reference them in this text. In many traditional IoT systems and applications, smart objects are stationary, and are built into the physical infrastructure of homes, offices, roads, etc.

The Internet of Mobile Things (IoMT) is a subset of IoT, where smart objects may be moved to different locations from time to time, or even move autonomously, but remain remotely accessible and controllable from anywhere on the internet [2]. IoMT systems commonly includes devices such as mobile phones, vehicles, robots, and wearables, all of which can have high connectivity and locomotion capabilities. Some examples of IoMT applications and use cases include smart cities, smart homes, environmental monitoring, health care, logistics and more.

Another important concept that is related to IoMT is Edge Computing, which refers to a distributed computing paradigm that brings computation and data storage closer to the source of where data is generated [7]. In traditional cloud computing models, data is sent to centralized servers for processing and analysis. In Edge Computing, computing processing power is moved to the *edge* of the network architecture, closer to where the data is generated or consumed. This approach enables faster response time, reduces latency, and uses network resources more efficiently.

If we think of the IoMT examples given above, smart objects are great use cases for how Edge Computing can be leveraged to create more flexible and resilient applications. Considering that on IoMT systems edge devices are always on the move, increasing their processing power allows computations to be made anywhere without relying on fast and stable internet connections. It also allows for real-time analysis and actions, which are crucial in time-sensitive applications like autonomous vehicles, industrial automation, or remote monitoring systems.

On the context of IoMT and Edge Computing, there are some applications that leverage both concepts to extend the capabilities of simple sensors through more powerful edge devices. Consider a scenario where we have thousands of very simple sensors, without considerable processing power, generating data. We may use a more powerful mobile device, to opportunistically discover and connect with those sensors when nearby, collecting data being generated and transmitting through the internet for further processing. This is a great example of both IoMT and Edge Computing concepts being used in practice, and it is very important for understanding the motivation and goals behind this project, since EdgeSec Framework was created and developed in the context of this type of system.

2.2

WPAN and Bluetooth Low Energy (BLE)

Personal Area Networks (PANs) are computer networks for interconnecting electronic devices within a person's workspace area [8], that is, within a short range of a few centimeters to a few meters of physical space. Wireless Personal Area Networks (WPANs) are PANs that transmit information over wireless network mediums and technologies. Some examples include Zigbee and Bluetooth, with the latter being the most widely used WPAN.

Bluetooth is a low-power, short-range radio technology that streams data over 79 different channels in the 2.4Ghz frequency band [9]. It can be used by stationery or mobile devices to exchange data over short distances. Bluetooth Low Energy (BLE) was introduced in version 4 of Bluetooth core specification, and it is a different technology designed for very low power operation [10]. This means that BLE consumes significantly less power than the original Bluetooth, allowing it to be used by devices with more strict energy requirements.

BLE communication can be established using different configurations and topologies, and there is a trade-off between energy consumption, latency, number of connected devices and data transfer rate. BLE architecture divides connected devices into Central devices, which are usually more powerful mobile phones, and Peripheral devices, which are small, low-power and resource constrained devices. A Peripheral device can only be connected to a single Central device at a time, while a Central device can keep multiple connections simultaneously. Figure 1 illustrates a connection topology [11].

Considering the roles that BLE defines for each device in a communication session, it is clear that this is a very suitable technology for the examples of IoMT and Edge Computing applications mentioned in the previous section. In fact, BLE is widely used in IoMT systems, where mobile devices act as the Central BLE device, and sensors act as the Peripheral BLE device. Moreover, BLE scanning properties and the relationship between Central and Peripheral devices are properties that make it a perfect technology to implement the opportunistic discovery of Smart Objects by Mobile-Hubs and Gateways.

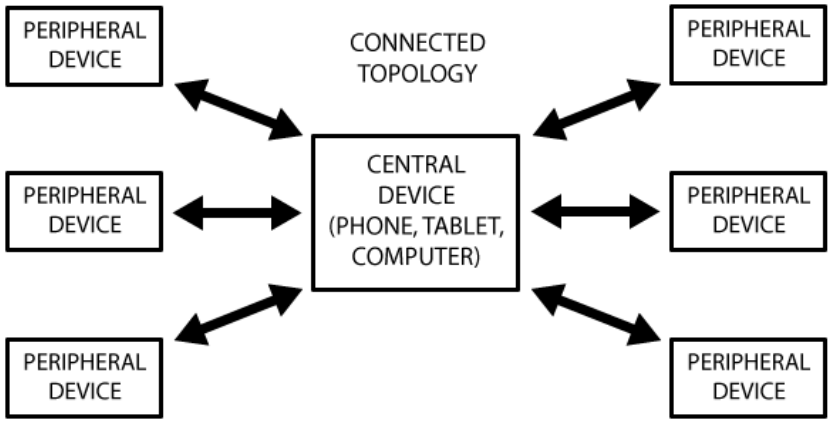

Figure 1 - Bluetooth Low Energy Connection Topology

2.3

Security Concepts

There are many different techniques and technologies that can be used to create security when users or devices are communicating and exchanging data in a network. In the context of this work, communication is done between devices such as servers, mobile phones, or smart objects and sensors. In the examples given below, we are only considering these type of devices as main actors,

In general, we are always aiming to enforce one or more basic security protections. The ones that are relevant in this project and are covered in some way by EdgeSec are:

- Authentication: there should be a way to verify if a device identity in a network is valid, and if it is really who it says it is. If a device tries to impersonate another by using a fake identity, a proper authentication mechanism should be able to identify this malicious activity and deny any type of access or action on the system.
- Authorization: there should be a way to verify if a certain device with a known identity is allowed to perform certain actions in a system. An example of authorization is the decision of whether two devices are allowed to communicate with each other. Another example is allowing or denying the permission for a certain device to access a certain type of data in a server.
- Confidentiality: when two devices are exchanging messages, there should be a way to avoid disclosure of the messages' content from unwanted third parties. This usually involves some type of encryption to transform the message in a way that only the two ends of the communication can decrypt and understand the message, making it unintelligible for others that capture the messages while it is being transferred.
- Integrity: there should be a way to verify if a message sent through a network between two devices was not tempered or modified by a third party. With confidentiality, we assure the encrypted message will protect the data from being exposed to malicious actors, but we cannot ensure they will not temper the message. Integrity checks create this type of protection by using some mechanism that allows us to verify if a message has been modified during transport.

In sections 2.3.1, 2.3.2 and 2.3.3, some of the mechanisms used in this project to achieve the security concepts listed above will be mentioned and briefly explained. This is not intended to be an extensive and complete explanation, as these are complex subjects that require a more in-depth analysis to be fully understood. The goal is to present and summarize some of these topics for readers that are not familiar with, which will make it easier to understand how and why they are used in EdgeSec.

2.3.1

Message Authentication Code (MAC)

Message Authentication Code (MAC) is a security mechanism used to create both authentication and integrity protections on a message [12]. This is done by

concatenating a short piece of information (the authentication code) at the end of the original message, which is used to verify if both the identity of the sender and the message content are valid.

These protections are achieved because the short piece of information added to the message must be generated with a security key that only the sender and receiver should possess. Because both sender and receiver use the exact same key, this is similar to symmetric encryption, which is going to be further explained in section 2.3.2.

The sender should use the security key to perform a transformation on the original message, generating a unique MAC. After receiving the payload, the receiver will separate the message from the MAC, perform the same transformation using the same key, and compare both MACs. If they match, it means the message was sent by the right sender because only they would have access to the key. And we can also assure the message integrity because any change or tampering on the message content would result in a different MAC.

In terms of implementation, the transformation mentioned before can be achieved through different security algorithms. One of the most common, and the one used in this work, is a cryptographic hashing function. Hashing functions are one-way, deterministic functions that transform an input of variable length into an output of a fixed size. Because it is a one-way function, there is no way to reverse it. In other words, it is very hard to guess the input based only on the output. Additionally, because it is deterministic, the same input will always generate the same output. And it´s rare, although possible, for two different inputs to generate the same output, an event known as *collision effect*. A good hash function shall avoid this to occur for inputs with a high correlation between them. For inputs that have a low correlation between them, it might happen but, usually the inputs will not belong to the same context, so we expect the system to reject the out of context input.

These characteristics make hashing functions an ideal implementation for Message Authentication Code algorithms. The input is usually the concatenation of the message with the security key. And the output is the MAC that is sent together with the message. Upon receipt, the receiver can concatenate the message with the key and perform the hash function again, which should result in the same MAC. From now on, we will use the acronym HMAC to denote hashing message authentication code algorithm. Figure 2 illustrates a Message Authentication Code algorithm using a hashing function.

In this project we use two different MAC algorithms to achieve authentication and integrity of messages in our proof-of-concept implementation: HMAC-MD5 and HMAC-SHA1. MD5 is a type of hashing function that, despite not being the most secure and presenting some vulnerabilities, was preferred due to lower computational requirements, allowing it to run smoothly on low power devices. It generates a 128-bit output and was designed by Ronald Rivest in 1991[13]. SHA1 is another very popular type of hashing function that produces a 160-bit output and was designed by the United States National Security Agency [14]. It is stronger than MD5, but still not recommended for production use today due to known vulnerabilities. These two algorithms were chosen because of the many available

open-source implementations for low power devices, which is not the case for more modern and advanced MAC algorithms.

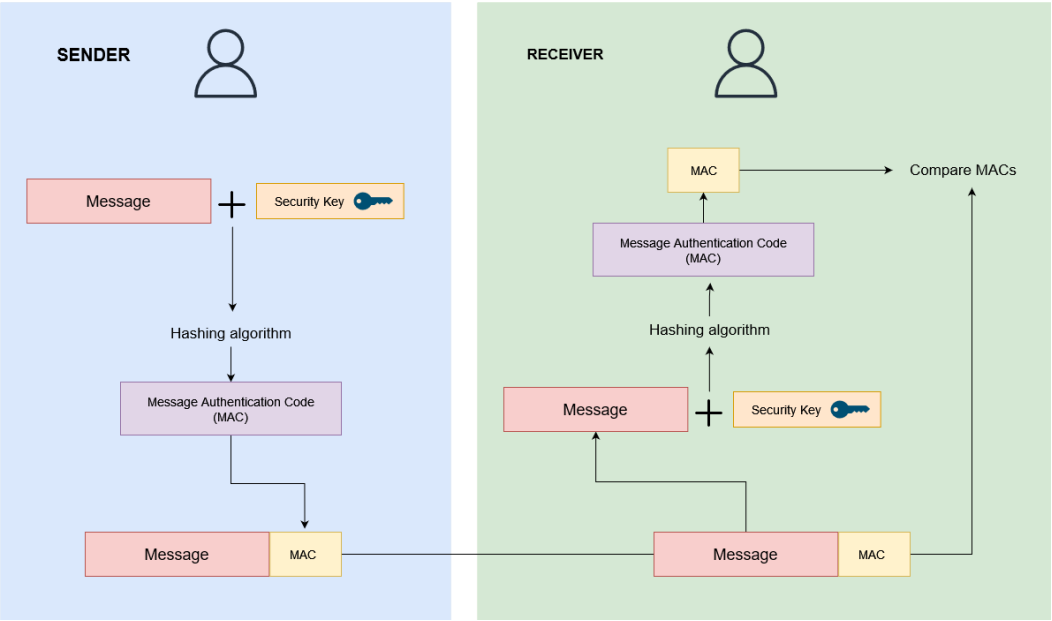

Figure 2 - Diagram of how Message Authentication code works

2.3.2

Symmetric Key Encryption

Data encryption is the foundation for protecting data on the internet. Because we exchange messages through the internet's public infrastructure on a global scale, a security mechanism to hide the content of what is being transferred is essential to many modern services, such as texting, online banking, e-commerce, and more. [15]

For this work, data encryption is used to achieve confidentiality in messages exchange between devices. More specifically, we use symmetric key cryptography, which is when the sender and receiver share the same cryptographic key. This same key is used to both transform plain text into cipher text and revert cipher text back to plain text. This contrasts with asymmetric cryptography, where there is a pair of public and private keys to generate unique secrets to each side of the communication.

There are two types of symmetric cryptography, with stream ciphers or block ciphers. In stream ciphers, the bytes are encrypted sequentially one by one. On block ciphers, blocks of bytes are encrypted as a whole one by one. Each algorithm uses a different number of bytes in each block. In this project, only stream cipher will be used.

The advantage of using symmetric key cryptography is because it is faster and lighter than asymmetric cryptography. This makes it way more accessible, allowing it to be used in simpler devices with smaller memory size and low processing power, such as IoT sensors. The most important disadvantage is the fact that both sender and receiver need to agree on a unique secret key, which can be complex when managing many devices communicating in the same system [16].

In our proof-of-concept implementation, we use RC4 as the symmetric key cryptography algorithm of choice. It is also not considered the most secure nowadays and presents many vulnerabilities. However, it is remarkably simple and fast, being a great choice for a POC with low-powered devices. It was also designed by Ronald Rivest, in 1987 [17]. Similarly to the MAC algorithms, RC4 was one of the few algorithms were we can easily find open source implementations for low-power devices. More advanced and secure algorithms lack this type of support.

2.3.3

One-time passwords (OTP)

Passwords and secrets are used across systems and algorithms to enforce data authentication, authorization, confidentiality, and integrity. Depending on the application and use case, always using the same password multiple times can be considered a vulnerability. Because of this there is a security mechanism called One-time password (OTP), where we only use a password for a single operation or transaction. A new password should be used for subsequent operations, avoiding attacks that replicate past states [18].

There are different types of OTP algorithms, including:

- Based on time-synchronization, where two parties synchronize their clocks and generate a unique code based on the current timestamp and a shared secret key.
- Based on hash-chains, were new OTP values are based on previous ones, creating a chain of passwords that only the two parties can agree on.
- Based on challenge-response, where one party should provide a response to a challenge from the other party.

OTPs are widely used in many internet services to establish two-factor authentication, where a user typically stores the shared key in a personal device and inputs a newly generated code every time it wants to re-authenticate [18].

In this project, we use OTPs to enhance the security of communication sessions between devices. Every time a Mobile-Hub and a smart object try to authenticate with each other, a new OTP is generated, ensuring that old communication sessions are invalid, and that new keys should be used.

2.4

ContextNet and Mobile-Hub

ContextNet is an IoMT Middleware that provides context services for wide and large-scale pervasive applications [6]. Its singular feature is that it employs mobile smartphones as hubs for discovering and connecting smart edge devices to the upstream servers on the internet. Some use cases are remote monitoring, coordination of a network of drones, management of Bluetooth sensors in modern hospitals and smart cities.

ContextNet's main component is the ContextNetCore, a network of servers running in the cloud that are responsible for providing the communication and context distribution capabilities. The Core is composed of many different processing servers that are used for different purposes, and cloud gateways that connect these inner processing nodes to external devices. It is extensible and flexible, allowing for new services and software modules to be implemented and integrated as new nodes inside the Core.

Mobile-Hub is another IoMT Middleware that runs on Android devices and is used as a mobile gateway and hub within the ContextNet architecture [3]. This Middleware serves as a bridge between simpler edge devices that do not have internet capabilities to communicate with ContextNetCore processing nodes and to send and to receive data over wide range internet protocols and local area protocols.

Mobile-Hub was also designed to be a flexible and extensible middleware. Its architecture facilitates the integration of new technologies for WPAN and WLAN communication. By leveraging the use of generic interfaces and implementing new drivers and custom configurations, we can easily introduce new devices to the system. New configurations can also be retrieved from upstream servers, without having to prepare the application for all types of devices that it may encounter. This means that Mobile-Hub can be extended dynamically, during runtime. A new version of Mobile-Hub, known as Mobile-Hub 2, introduced a new code architecture and technology stack. Most of the mentioning to Mobile-Hub in this document will refer to version 2.

The reason for citing these two Middlewares as background is because they served as a motivation for creating the EdgeSec framework. Despite the conceptual idea behind the framework being generic and flexible, ContextNet and Mobile-Hub were used as test cases for developing and proving the usability of this security solution, as we are going to show on the proof-of-concept implementation chapter. These Middlewares represent the type of application that EdgeSec framework aims at supporting and will be used as example throughout this document to illustrate some of its features.

3

Related Work

As expected, due to the widespread use of IoT systems and the importance of security in today's connected world, there is a considerable amount of related work. This is especially true if we analyze security in a broad way, including server-side security, client-side security, network security, hardware security, and more. To narrow down the scope of the works related to this one, we are going to focus solely on security frameworks aimed at IoT systems. Other types of IoT security solutions that are not designed as frameworks will not be considered.

In my research for related work, I observed a pattern for types of security framework solutions that could be separated into two kinds of approaches: *frameworks for detection and modelling of security threats*, and *frameworks for prevention against security attacks*. Apart from some specific exceptions, most of the work I found could fit into one of these two groups. Each group of solution target different ways in which an IoT system can be protected from malicious activities.

For the first group, it is common to see an intersection between security and Machine Learning techniques. This type of solution uses Artificial Intelligence strategies to detect and model viruses and other types of attacks.

An example of this kind of solution was proposed in the "IoT Security Framework for Smart Cyber Infrastructures" paper, where the authors present a security framework for IoT applications in smart infrastructures, such as smart homes and smart buildings [19]. The framework utilizes intrusion detection systems (IDS) to continuously monitor the network and collect data from sensors in order to detect any unusual activity within the IoT environment. This data is used to identify specific sensors and compare their behavior to expected patterns. The framework categorizes any attack that gets detected based on the type of abnormal behavior, taking appropriate recovery measures, such as re-authenticating the sensor, discarding sensor data, or modifying the network configuration.

There is another example that builds on top of the work just mentioned, and it follows the same paradigm of using AI to detect knowledge-based and anomaly-based attacks [20]. It has a strong focus in using Software Defined Networks (SDNs) and Network Function Virtualization (NFV) technologies to increase control over the IoT network and enhance the ability of detecting and reacting to attacks.

It also proposes a security framework in which IoT devices are part of a network architecture controlled by SDNs/NFVs controllers that actively monitor the activity using Machine Learning algorithms to detect anomalies and launch mitigation actions in a closed loop.

The two frameworks mentioned above use a reactive approach, detecting and acting on attacks that are already under way. The other type of related work, which is the one where EdgeSec fits in, is usually associated with a software tool, service, or protocol that can be integrated into an IoT system architecture to prevent exploits from happening in the first place. It is a strategy of creating safeguards that ensure

authenticity, integrity and/or confidentiality to data, with the cost of introducing some overhead and additional processing.

The paper written by S. Sridhar is an example of this second type of related work, which is very similar to EdgeSec in terms of general concept and structure, considers an architecture of edge devices, mobile gateways, and cloud servers, and it uses encryption and session keys to protect all data in transit [21]. It requires devices to be previously registered in a centralized database, or Master Key Repository, to securely authenticate elements of the communication. This creates a protection against malicious devices that try to impersonate others and send fake messages.

However, it differs from EdgeSec in a few ways. It uses asymmetric cryptography algorithms, which cannot be supported by some types of devices, and does not verify data integrity on every message exchanged. Most importantly, it does not provide full flexibility to change the cryptographic algorithms.

The work done by R. Hsu is the most closely related to EdgeSec in terms of goals, scope, and security strategy [22]. It also recognizes the fact that many related projects only offer basic security protection, and fail to overcome the challenges of device heterogeneity, key management complexity, and computational power scarcity on the same IoT system.

This work addresses these issues by proposing a reconfigurable framework that focuses on edge computing, called *Reconfigurable Security Framework for IoT (ReSIoT)*. This is done through the introduction of a security agent (SA), which is an edge device with more processing power dedicated to handle cryptographic algorithms and reduce the computation cost on other edge devices. These other edge devices only need to keep a security key to communicate with the SA.

The SA is responsible for managing and distributing keys to nearby devices through a global key management system, and it inherits the protection mechanisms of the underlying protocol communication layer. The goal is to ensure the basic security requirements of confidentiality, integrity, availability, and non-repudiation. These are classified as security functions (SFs), which are handled solely by SAs.

The ReSIoT architecture includes three main layers: (i) connectivity, consisting of network protocols such as BLE, UDP, MQTT, and TLS; (ii) security and resource layer, which is where the SA sits, and where it performs SFs; and (iii) application layer, consisting of application resources on server and client level. They also define Reconfigurable Security Functions (RSFs), which is a protocol that can be thought of as a middleware used to perform the security algorithms and fulfill the SFs.

Although ReSIoT solves many of the issues that are open on other works described here, there are still some scenarios where EdgeSec can offer a better solution. In particular, adding an extra device to work as a security agent is not practical or even possible in many IoT Systems, such as applications that require a very large number of edge devices distributed across a wide area. Another important consideration is that ReSIoT does not use a concrete implementation for RSFs, which can vary a lot depending on the device and operating system it is running on, as well as which security operation is required in each communication.

Section 4 shows that EdgeSec differs from that approach by ensuring the same security requirements without an extra device to run as security agent. We aim for a specific type of IoT architecture, where a mobile gateway is always present in the edge near sensors. We can leverage these mobile gateways to serve as main security agents, with the advantage of its mobility to cover wide areas and a large number of sensors. Additionally, EdgeSec provides a solid implementation of its core capabilities, and proposes new security mechanisms that consider computational power limitations, so that these operations can be performed end to end, including on edge sensors, mobile gateways, and cloud servers.

4

EdgeSec Framework

Before explaining how EdgeSec framework works, it is important to go through how it was conceived and the foundations behind it. The general architecture and security algorithm used by the framework was first proposed in [5]. In this work, the authors created a security solution aimed at IoT systems, with special focus on edge devices and edge computing.

The main goal of this solution is to achieve data integrity, authentication, confidentiality, and access control on a decentralized and heterogeneous IoT network. The architecture considers a system of three main components: (i) processing servers on the cloud; (ii) mobile devices acting as gateways; and (iii) edge devices or sensors that generate raw data. A security protocol for establishing a secure connection between mobile gateways and edge devices is presented as part of the solution.

The authors also describe a threat model for this type of architecture and IoT environment. They arrange threats into two distinct groups: threats to the operation of entities of the IoT system; and threats to the communication between entities of the IoT system. We can assume the same threat model for EdgeSec Framework, and more details about each threat group can be found in [5].

Both the architecture and the protocol were proposed conceptually and made concrete as an extension of ContextNet and Mobile-Hub. These middlewares were used as examples and use cases of how this solution could improve the overall security of an IoT system.

To prove the concept, a few implementation projects were developed with the goal of turning this into a functional solution that could be effectively used in real world applications. One of these projects, named EdgeSec, consisted in implementing the security protocol inside ContextNet and Mobile-Hub, together with a microcontroller acting as an edge device [6]. It was executed and tested end to end, achieving real and practical results. Later, a new implementation project adapted EdgeSec to Mobile-Hub version 2, improving the performance of the security algorithm.

Despite being successful in proving that the solution could create a great level of security for these middlewares, there were a few problems and challenges that still needed to be addressed in this EdgeSec proof-of-concept implementation. The most important was regarding flexibility: the code was protocol specific, and would only work with BLE as WPAN protocol, HMAC-MD5 as authentication and integrity mechanism, and RC4 as cryptographic algorithm. Any change in one of these protocols would mean a complete rewrite of the project's code. Additionally, the edge device implementation was too hardware-specific, so that any new device that is to be used within the architecture also required a complete rewrite of parts of the code.

In order to address these problems, we develop a new solution, proposed here in this work, called EdgeSec framework. The framework aims to solve the flexibility problem and achieve a more robust security solution for the ContextNet and Mobile-Hub 2 middlewares and can be considered an evolution of the original

EdgeSec implementation. Because of this, EdgeSec framework shares the same foundations of the initial conceptual proposal, including the general architecture and security protocol. In section 4.1, we will explain in detail how the existing architecture and protocol works, with minor improvements and changes. In section 4.2, we will explore the main contribution of this project, and how the framework expanded previous works into a more complete solution.

4.1

EdgeSec Architecture and Protocol

4.1.1

Initial Set up

The architecture can be divided into three components:

i. Edge devices or sensors acting as Smart Objects;

ii. Mobile devices acting as bridges or mobile gateways;

iii. A central processing node acting as an authorization server.

Prior to using EdgeSec, a few data initialization needs to take place:

- The authorization server must store the ID of all Smart Objects that are part of the system.
- The server should also hold a relation of all mobile gateway/smart object pair that are allowed to authenticate and communicate with each other. This is needed to enforce any required access control.
- Each Smart Object must store three security keys: two symmetric authentication keys, one of them being its own key and the other being the server`s key, and an encryption key.
- The authorization server must store its own key along with the authentication and encryption key of all registered devices.

These steps can be set up during a registration process, where each device that will be part of the system is registered and configured with all keys and data necessary. This process can be manual or automated. There are plans to create an automation mechanism, where a batch of devices get all registered at once, facilitating the use of EdgeSec in large scale systems. This is one of the goals as a future work for this project. The initial set up is illustrated in Figure 3.

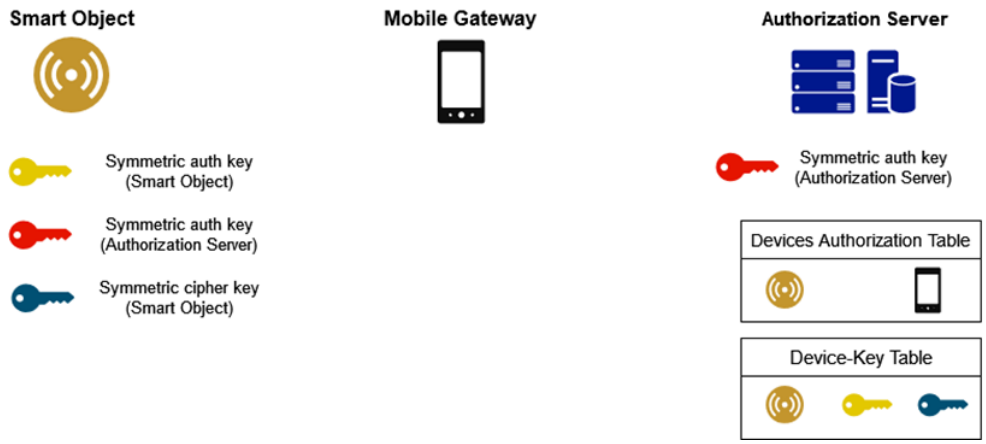

Figure 3 - Three components of EdgeSec architecture and its initial set up data

4.1.2

First connection and handshake

EdgeSec comes into action when a Mobile Gateway finds a nearby Smart Object and tries to communicate with it securely. When these two devices start the process of communication, the security algorithm starts. This algorithm can be further divided into four parts: handshake, authorization, authentication, and secure data exchange.

Because most WPAN protocols do not support application-level data being exchanged before a connection is already established, this handshake happens after a first connection. However, despite being connected, the devices can only exchange real sensor data after the authentication process is finished. If this process fails to be completed, the connection is immediately terminated.

The first part of the communication process begins with a handshake between the Mobile Gateway and Smart Object, where they exchange a few messages to share their IDs with each other.

In the original security solution, this handshake restricted the two parties to just sharing their IDs. EdgeSec Framework redesigns and extends the handshake into a more flexible process, allowing more data to be exchanged, such as framework version and protocol suite negotiation.

Although the suite negotiation is not strictly needed for the functionating of the algorithm, it significantly improves its flexibility, allowing for devices to decide dynamically during runtime which protocols are going to be used in the secure communication. This negotiation should be similar to how TLS (Transport Layer Security) negotiates ciphers. Because the implementation of protocol handshakes can be difficult, an alternative is to store all the protocol suites supported by Smart Object and Mobile Gateway during registration phase. This way, the Authorization Server can decide which protocols to use based on the suites supported by each device, simplifying the handshake process.

After connecting to Smart Object, exchanging IDs, and deciding on which protocols are going to be used, Mobile Gateway starts a connection with the Authorization Server through the internet using some type of well-known security layer, such as TLS or any VPN protocol. This security layer is important to make

sure that data is secured end to end, in all parts of the IoT architecture. It usually involves key exchange algorithms, digital signing, and digital certificates, as in TLS protocol. Since there are many reliable, modern, and proven solutions for this type of security layer, it is out of scope for EdgeSec Framework project.

4.1.3 Authorization

In this step, the Mobile Gateway sends the handshake information (pair of IDs and negotiated protocol suite) to the Authorization Server to ensure that it can go forward with this communication. The Server should check in its previously configured database if the two parties are allowed to communicate. If the result is positive, the server starts to prepare a response that contains a few different elements. First, it needs to generate an OTP (One Time Password), which is unique to each authentication process, with the following equation:

$$OTP = hash(OTPChallenge + Smart\ Object\ ID + Mobile\ Gateway\ ID + Smart\ Object\ Authentication\ Key) \quad (1)$$

Where *hash* can be any secure hashing function, and *OTPChallenge* is a pseudo random 13-bytes positive number.

Another element generated by the server is a random Session Key. After authentication, when data is being transferred between Mobile Gateway and Smart Object, the OTP is used as a HMAC signing key to prove the authenticity and integrity of all messages, and the Session Key is used as a symmetric cryptographic key to encrypt and decrypt the message content. Figure 4 illustrates this process of protecting data in EdgeSec.

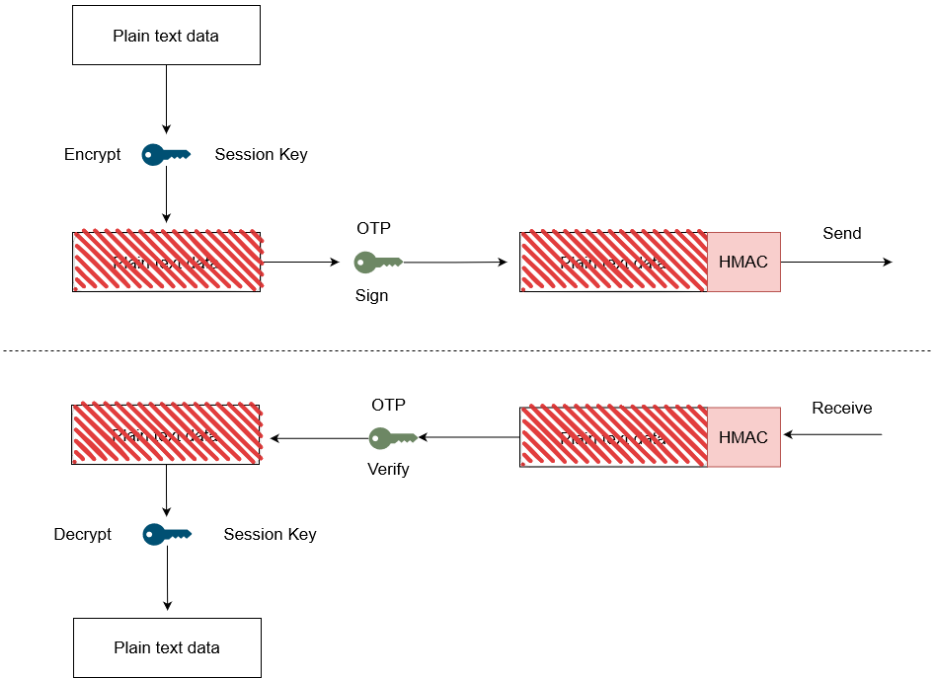

Figure 4: Process of securing a message using the Session Key to encrypt/decrypt (confidentiality) and OTP to authenticate (authenticity and integrity).

The server responds to the authorization request by sending back the OTP, the Session Key, and an authentication package, named *PackageK*. This package will later be used by the Smart Object to verify messages, and it follows the equations below:

$$PackageK = encrypted(OTPChallenge + Session\ Key) \quad (2)$$

$$Authorization\ Response = OTP + Session\ Key + PackageK + HMAC(PackageK) \quad (3)$$

Where the encryption key in (2) is the Smart Object Symmetric Encryption Key, and the HMAC key in (3) is the Authorization Server`s Authentication Key. The whole authorization process is illustrated in Figure 5.

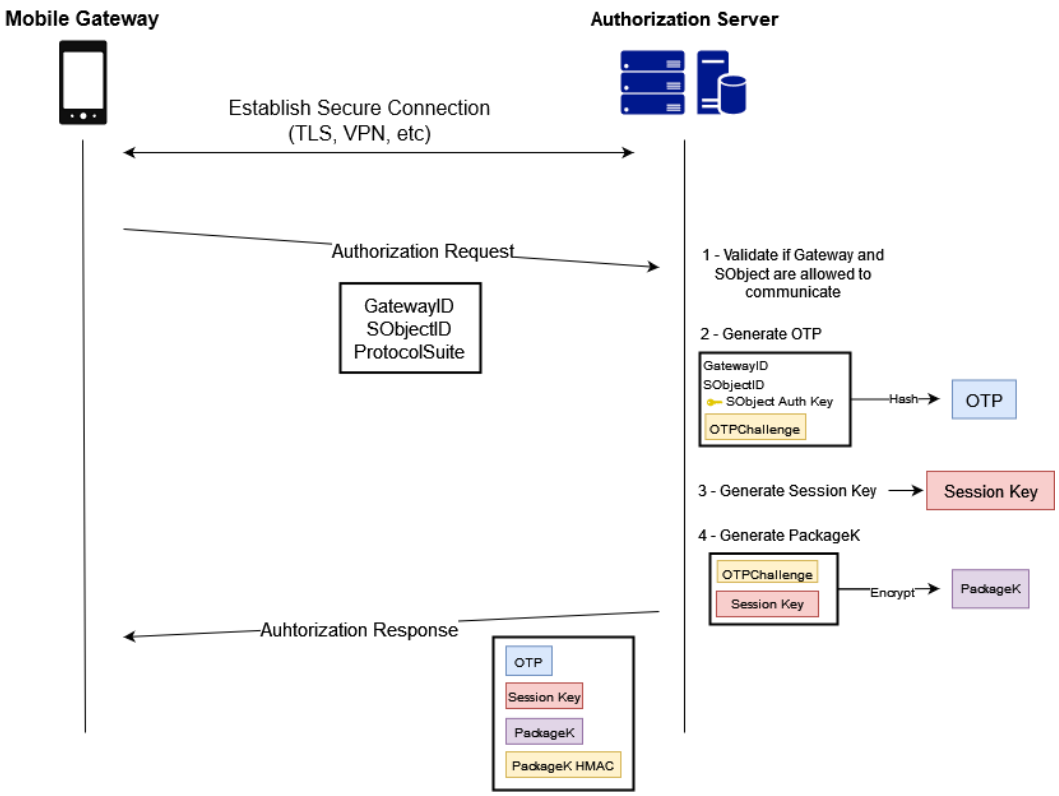

Figure 5 - Authorization process in EdgeSec

4.1.4

Authentication

After receiving a positive authorization response, the Mobile Gateway is ready to proceed connecting with the Smart Object. To notify the Smart Object of the authentication process, and prove its identity, a *HelloMessage* is sent, which has the following format:

$$HelloMessageContent = PackageK + HMAC(PackageK) \quad (4)$$

$$HelloMessage = HelloMessageContent + HMAC(HelloMessageContent) \quad (5)$$

Where the HMAC key in (5) is the *OTP* from the authorization response.

After receiving this message, the Smart Object starts to validate the authentication process. For this validation, it needs to re-generate the HMACs using keys previously stored in its memory and compare with the HMAC received from the Mobile Gateway. If both the *PackageK* HMAC and *HelloMessage* HMAC match the expected values, the message passes the integrity and authenticity test. The Smart Object then extracts the *OTPChallenge* and *Session Key* values from *PackageK* by decrypting it, and then re-generates the *OTP*. Now, it can store both

the *OTP* and *Session Key* in its memory, which are going to be used to encrypt, sign, and validate future messages.

To finish the authentication process and signal to the Mobile Gateway about the success in validating the message, the Smart Object responds with a signed *HelloAcceptedMessage*, following the equation below:

$$HelloAcceptedMessageContent = Gateway\ ID + Smart\ Object\ ID \quad (6)$$

$$HelloAcceptedMessage = HelloAcceptedMessageContent + HMAC(HelloAcceptedMessageContent) \quad (7)$$

Where the *OTP* is the HMAC key in (7).

The Mobile Gateway receives this message, validates the signature, and if successful, the authentication process is over. Figure 6 illustrates the authentication process.

Now, both devices communicate securely, encrypting messages with the *Session Key* and signing them with *OTP*. A summary of all messages exchanged during the authorization and authentication processes can be seen in Figure 7.

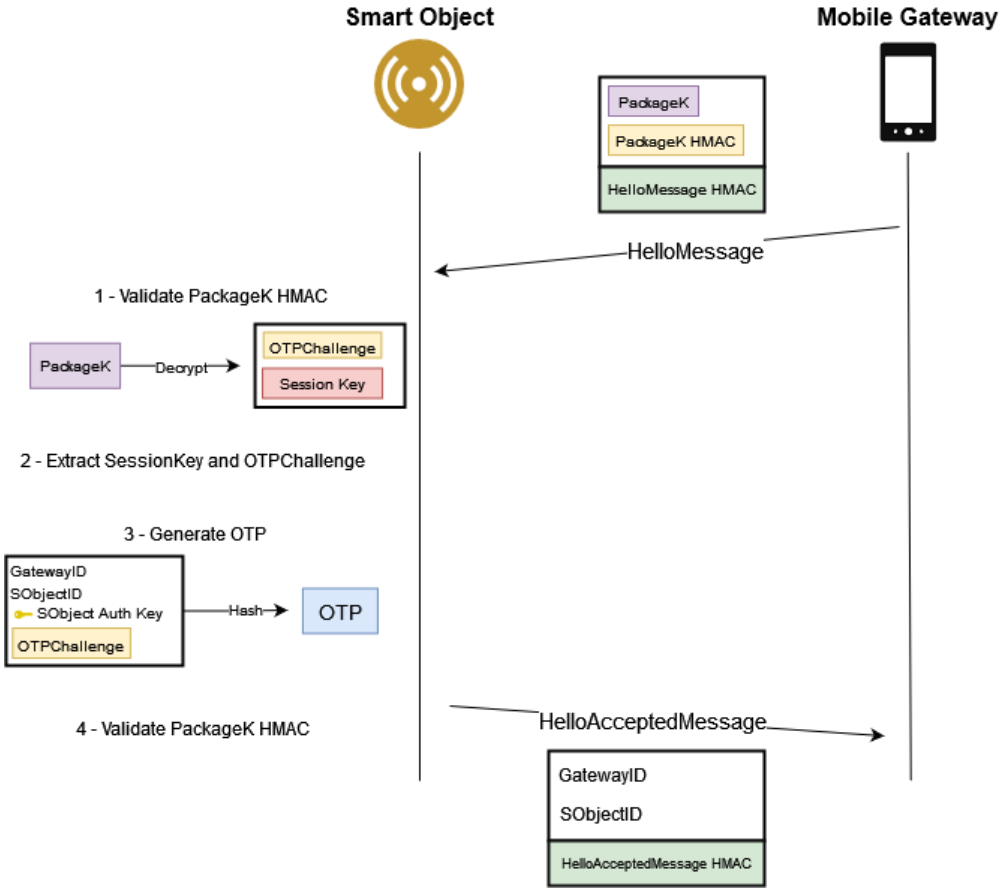

Figure 6 - Authentication process

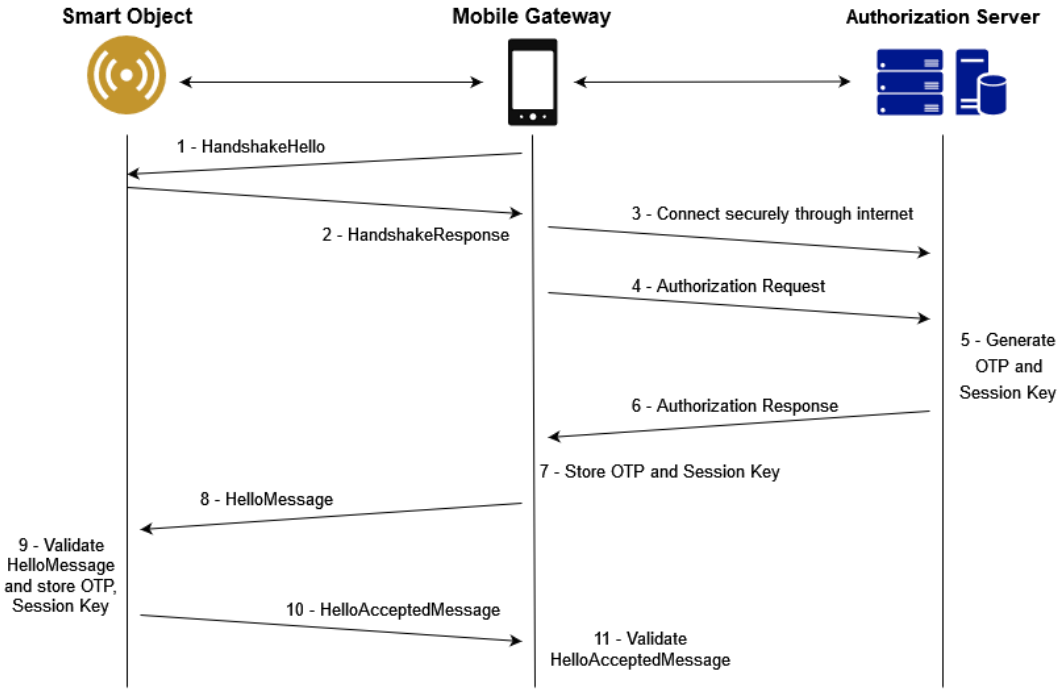

Figure 7: Messages exchanged between components during authentication and authorization processes.

4.2

Framework

EdgeSec Framework expands and evolves the original EdgeSec architecture and protocol described within section 4.1. To achieve the desired flexibility, some logical changes were needed on the protocol algorithm, and some new concepts were introduced in the architecture.

4.2.1

Objectives and Use cases

The goal of the framework is to be a flexible and extensible security solution for ContextNet and Mobile-Hub middlewares, allowing the use of different protocols and algorithms in communications between devices. These middlewares are usually used in different types of IoT applications, and many would benefit from having an improved level of security such as the one provided by EdgeSec framework. Some examples of use cases are:

- Smart Cities, where sensors are distributed across a vast area, and data is collected through mobile phones that move throughout the city. This might include sensitive and private data, so the framework could ensure privacy and protection to users and administrators.
- Smart Hospitals, where different sensors can be spread around a hospital facility, collecting data about patients and doctors. Because medical records are always sensitive information, the framework could be integrated to a smart hospital application to ensure data security.
- Industry 4.0, where many sensors can be installed around the industrial plant and integrated with the machinery. This type of application can be critical because edge devices can sometimes control the operation of very important machines and processes. If a malicious hacker gets control of these assets, an attack could cause great economic and physical impact. EdgeSec Framework could be integrated into the software that controls the IoT sensors to ensure the safety needed and protect against attacks.

From these examples, we can see that the framework should allow integration with a wide range of IoT systems and to be compatible with devices used in different sectors. To enable this flexibility, the framework is designed to accept different types of plugins that can be provided by the user for the framework during initialization. Details of how a plugin works are going to be given in section 4.2.2.

Considering the use cases mentioned above, we can define two types of target users for the framework:

- IoT application developers, who want to enhance security in their applications without the need of specific security knowledge. These developers can just import, instantiate, and initialize the framework along with the desired plugins.
- IoT devices manufacturers or hardware developers, who want to make their devices more broadly compatible by developing plugins for the protocols supported by these devices.

4.2.2

How it works

To enable the extensibility and flexibility mentioned before, the framework design is divided into three main components:

- **Framework core**: responsible for executing the main algorithm to create the security mechanisms. It runs on Mobile-Hub, which is connected both to the cloud servers and to Smart Objects and have a larger processing power.
- **Framework authorization server**: responsible for storing the cryptographic keys and registration of each supported device. It receives authorization requests from the mobile gateways, authorizing or blocking a connection. It runs on a processing server in the cloud.
- **Protocol specific plugins**: plugins that implement interfaces defined by the framework to ensure a specific protocol can be used by the core in its algorithms. These plugins are divided into three types: authentication, cryptography, and transport.

The framework core runs the same protocol that was proposed in the original architecture, and explained in section 4.1, with the difference that it generalizes and abstracts all protocol specific parts into external plugin function calls, and it adds a handshake process to allow cipher negotiation. This negotiation was designed to be similar to a standard TLS handshake, and it is described in a high level below:

1. Mobile-Hub, acting as client, sends the *HandshakeHello* message containing its ID, version of the Framework, and a list of which protocol suites it supports, in a preferred order. The suites supported by Mobile-Hub will depend on the plugins available during initialization of EdgeSec framework.
2. Smart Object, acting as server, sends the *HandshakeResponse* as response, containing its ID, and the selected protocol suite. If it does not support any of the protocol suites, it ends the connection.
3. Mobile-Hub receives and parses the response, storing the selected protocol suite that should be used.

The authorization server should also be connected to a database to store essential information used during authorization and authentication processes. This information includes:

- List of each smart device registered in the application.
- A secret private key known only to the authorization server and used to sign data.
- List of authentication keys for each of the registered devices.
- List of encryption keys for each of the registered devices.
- List of mobile gateways that are authorized to connect to each of the registered devices.

- List of all protocols supported by the plugins used in the application.

Upon receiving an authorization request through the internet, the server will fetch the required information from this database and perform some authorization checks. It then prepares a response to the mobile gateway, including the keys and other values that are used later during authentication and connection session. This is the process described in section 4.1.3.

Lastly, plugins should implement one of the three interfaces provided by the core:

- **ITransportPlugin**: interface for plugins that implement a WPAN communication protocol, such as BLE, Bluetooth Classic, and Zigbee.
- **IAuthenticationPlugin**: interface for plugins that implement cryptographic hashing function and digital signing algorithms, such as HMAC-MD5, HMAC-SHA1, and DSA-SHA256.
- **ICryptographicPlugin**: interface for plugins that implement symmetric encryption algorithms, such as RC4, AES, DES.

Each of these interfaces define functions that are used by the Framework Core during the EdgeSec security algorithm. These functions are strongly tied to how the algorithm works, and the steps taken to authorize and authenticate a device. However, they are generic, and can be implemented using different protocols and techniques. Each plugin should have an ID that uniquely identifies the protocol implemented by it. A protocol suite is a string code composed of the three plugin IDs for each protocol concatenated together. An example for a suite that uses BLE as TransportPlugin, HMAC-MD5 as AuthenticationPlugin and RC4 and CryptographicPlugin would be "*BLE_HMAC-MD5_RC4*".

In the following sections we present code snippets detailing the content of each interface, with brief comments for each function. These interfaces are written in Kotlin programming language because a test implementation was made in Kotlin. However, some language-specific details can be ignored, and the same interfaces can be reproduced in any other programming language.

4.2.3

ITransportPlugin

The transport interface has many references to *Observable* keywords. This is a data type available in Kotlin to make use of the Observable design pattern. These data types represent an asynchronous emitter, an object that emits a certain primitive value continuously in an asynchronous way, which is the usual behavior of a communication protocol.

```
interface ITransportPlugin {

   /* Returns ID of protocol implemented by plugin */
   fun getProtocolID(): String;

   /* Scan for nearby compatible devices using a transport protocol */
   fun scanDevices(): Observable<String>;

   /* Tries to connect with a device using a transport protocol */
   fun connect(deviceID: String): Observable <Boolean>;

   /* Verifies if device is compatible with EdgeSec framework */
   fun verifyDeviceCompatibility(deviceID: String): Observable <Boolean>;

   /* Sends the handshakeHelloMessage to device */
   fun sendHandshakeHello(deviceID: String, data: ByteArray): Observable <Boolean>;

   /* Reads the handshakeResponse from device */
   fun readHandshakeResponse(deviceID: String): Observable <ByteArray>;

   /* Sends hello message to device */
   fun sendHelloMessage(deviceID: String, data: ByteArray): Observable <Boolean>;

   /* Reads the HelloMessageResponse from device*/
   fun readHelloMessageResponse(deviceID: String): Observable <ByteArray>;

   /* Reads data from device */
   fun readData(deviceID: String): Observable <ByteArray>;

   /* Writes data to device */
   fun writeData(deviceID: String, data: ByteArray): Single<Boolean>;

   /* Terminates connection with device */
   fun disconnect(deviceID: String);
}
```

4.2.4

IAuthenticationPlugin

```
interface IAuthenticationPlugin {

   /* Returns ID of protocol implemented by plugin */
   fun getProtocolID(): String;

   /* Digitally sign data using provided key with the protocol implemented by plugin */
   fun getMACSignature(data: ByteArray, key: Key): ByteArray;

   /* Verify a message authentication code signature */
   fun verifyMACSignature(data: ByteArray, key: Key, signature: ByteArray): Boolean;

   /* Generate a hash value using hashing function implemented by plugin */
   fun generateHash(payload: ByteArray): ByteArray;

   /* Return size in bytes of the hash generated by hashing function of protocol
implemented by plugin */
   fun getHashSize(): Int;
}
```

4.2.5

ICryptographicPlugin

```
interface ICryptographicPlugin {

   /* Returns ID of protocol implemented by plugin */
   fun getProtocolID(): String;

   /* Generate a random token using protocol implemented by plugin */
   fun generateSecureRandomToken(size: Int): ByteArray;

   /* Generate a secret key */
   fun generateSecretKey(seed: ByteArray): Key;

   /* Encrypt data using a provided key */
   fun encrypt(plainText: ByteArray, key: ByteArray): ByteArray;

   /* Decrypt data using a provided key */
   fun decrypt(cipher: ByteArray, key: ByteArray): ByteArray;

   /* Return size in bytes of the secret key of protocol implemented by plugin */
   fun getSecretKeySize(): Int;

}
```

5

Proof of Concept Implementation

One of the most important aspects of the proposed framework is proving its utility in real world applications and show it working in practice. This framework expanded from the original EdgeSec architecture to allow for more devices and protocols to be used. Because of this, it was essential that a few practical tests and prototypes were created to compare this framework architecture with previous solutions and ensure the project could achieve its goals. In this section we explain how the proof of concept was implemented, and in section 6 we go through the tests that were made.

5.1

Technologies, Hardware, and Environment

As already explained in section 4, the initial security solution that was base for EdgeSec framework was made concrete as an extension of ContextNet and Mobile-Hub middlewares. Mobile-Hub 2 runs on Android devices and is written in Kotlin programming language. Naturally, this first prototype of EdgeSec framework was also written in Kotlin and developed as an Android Library to facilitate its integration with the middleware.

Mobile-Hub 2 architecture is divided into different layers and services. On the base layer, it's the Core Library, which includes three main modules: (i) S2PA Gateway, responsible for managing connection with sensors through WPAN technologies, including collecting and writing data; (ii) Connection Gateway, responsible for managing connection with upstream servers through WLAN technologies, including sending sensor data to the cloud for processing; and (iii) MEPA Gateway, responsible for processing complex events flowing through the Middleware. On top of the core library is the Mobile-Hub Service, which bridges these gateways and its functionalities with the top-level layer. And in the top-level layer it's the Mobile-Hub Configurator, responsible for configuring and integrating Mobile-Hub with application layer apps.

Because EdgeSec Framework focuses on creating security mechanisms on WPAN communication, its integration with Mobile-Hub 2 was done inside the S2PA Gateway module. S2PA provides a WPAN programming interface for interacting with sensors through different WPAN technologies in a transparent way. To use a new protocol, we just need to implement the interface and import it inside S2PA configuration.

This was the method used for incorporating EdgeSec Framework. We implemented the S2PA's WPAN interface in a new module called EdgeSec-WPAN. Inside this module, the Framework code and all its plugins were initialized, creating an abstraction layer between S2PA and real WPAN technologies such as BLE. Figure 8 shows the architecture of Mobile-Hub 2, including the integration of EdgeSec Framework.

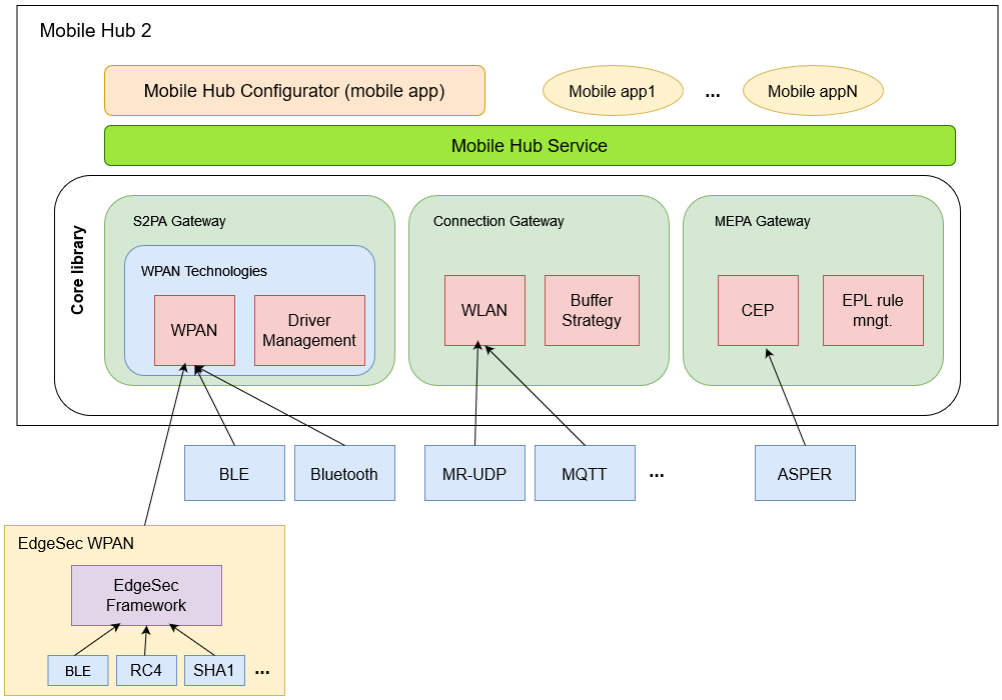

Figure 8 - Mobile-Hub 2 architecture + EdgeSec Framework integration

The prototype code was divided into three different parts, following the same logic division of the framework concept: core, plugins, and authorization server. For the core, a Kotlin library consisting of a few different modules was developed. The intention was to create a single deliverable file, such as a Java Archive (JAR), that can be imported and incorporated inside another Android project.

For plugins, the same idea was applied, where different libraries were implemented for each of the different types of plugins. In this case, because there was a prior implementation of EdgeSec using BLE, MD5 and RC4 algorithms, these same protocols were used to implement each plugin. An additional plugin was implemented using SHA1 to prove the use of multiple protocols and compare results.

As for the authorization server, it was developed as a separate Kotlin module, that ideally should be executed in a remote server that would be connected to the framework core code through a network. However, to simplify the development and facilitate testing, this code was implemented as an isolated module, also running on an Android device.

Figure 9 shows a class diagram of the prototype implementation. In Appendix I, each part of the diagram is shown separately to help understanding the different modules in more detail.

After the implementation of the framework, the next step was to incorporate and use it inside Mobile-Hub. A few small adaptations were needed to make EdgeSec Framework compatible with Mobile-Hub's WPAN interface. JARs for the Framework Core and each Plugin were imported and instantiated directly inside the middleware code. References for each plugin implementation were provided for the framework as reference during initialization and set up.

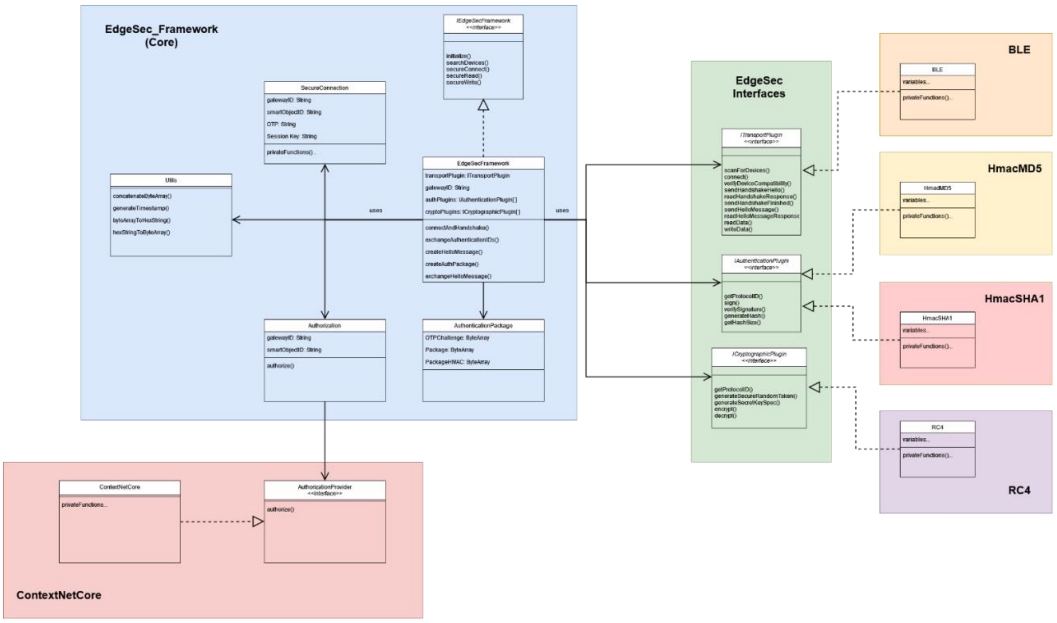

Figure 9 - Class Diagram of the implementation prototype

To run EdgeSec end to end, an edge device acting as Smart Object was also needed. For this, we used an ESP32 microcontroller, that has built-in Bluetooth and BLE capabilities. The framework and protocol logic were implemented in C++ language and uploaded to the ESP32 memory through Arduino IDE software. We used third-party C++ libraries to execute the MD5 and RC4 algorithms. For MD5, *ArduinoMD5* from Spaniakos [23] and *Hash-Library* from Stephan Brumme [24]. For RC4 and BLE, we used libraries provided by ESP32's manufacturer EXPRESSIF [25]. And for SHA1, *tiny-HMAC-c* library by kokke [26] was used.

A few different design patterns were used to create the implementation project and architecture. The main one is the Observer pattern, which was mostly used in the transport plugin and in the framework core. This pattern is very well suited to handle asynchronous communication and methods, which is important for many transport protocols, such as BLE. This type of message is used during authentication and secure data exchange operations between the mobile gateway and Smart Object. Because of this, the Observable data type, which is provided by Android's *io.reactivex* library, was used to define parameters and return types in the ITransportPlugin interface.

5.2

Practical Challenges

One of the most important concepts of the proposed framework is the flexibility of using different security protocols and algorithms in a dynamic way. As already described in section 4.1, the handshake process provides a way for two parties to negotiate the suite of protocols that are going to be used in the communication session.

However, in the prototype implementation, it was not possible to achieve a working solution for the handshake negotiation within the project's deadline. This

is a complex implementation, where some additional information such as the number of bytes for each protocol suite needs to be sent and parsed in a dynamic way. Despite doing some basic tests and designing the negotiation as pseudo-code, we could not have a full implementation in time.

As mentioned in section 4.1, the protocol negotiation is not necessary for the functioning of the security solution, and it was left as an opportunity for future work. As a workaround, it was decided to store the protocol suite supported by each device in the Authorization server, together with other registration data. During the authorization process, the server can select a protocol suite during runtime and return it to the Mobile Gateway as part of the authorization response. Because of this, during the Handshake messages, only IDs need to be exchanged, facilitating the implementation.

Another practical challenge faced in the prototype implementation was the development of the Authorization Server itself. EdgeSec architecture expects this component to be a server in ContextNet core, that can be connected to Mobile-Hub securely through the internet. Achieving this meant setting up a new processing server, developing code specific to this type of server, setting up a gateway on ContextNet core, and establishing a secure VPN or TLS connection.

All these steps were not the focus of this project and are not required to make EdgeSec framework work. Again, to facilitate implementation of this proof-of-concept, we simulated the authorization server as an isolated Kotlin Module running inside Mobile-Hub. This simplified the infrastructure set up and communication with the Framework Core.

6

Experimental Results

6.1

Performance tests

Because EdgeSec Framework includes a new architecture, a refactoring, and a redesign of the initial security solution, we decided to run performance tests on our proof-of-concept implementation. The goal was to measure how these changes would impact previous implementations, possibly gaining performance through code optimizations, and ensuring it could remain as a viable real-world solution.

By introducing security mechanisms on top of existing communications, it is impossible to completely avoid some performance overhead. The nature of many security operations, such as digital signatures and data encrypting, implies additional computation, that necessarily increases execution time. However, this performance overhead should be proportional to the improvements achieved with higher security and cannot create a considerable negative impact in the data flow or application behavior.

To evaluate the performance impact, a detailed profiling of every step in EdgeSec' s algorithm was made. This included measuring the time taken to perform each step of the authentication, authorization, and secure data exchange operations. These measurements were made using devices with the following hardware:

- Running **Mobile-Hub 2** code:
    - Android 11 Smartphone device
    - 2.3 GHz Octa Core CPU
    - 6GB RAM
    - Bluetooth 5.0
- Running the **Smart Object** code:
    - ESP32 Microcontroller
    - CPU Xtensa® Dual-Core 32-bit LX6,
    - 520 KB RAM
    - 240MHz Clock
    - Bluetooth 4.2

As these tests were made on the prototype implementation described before, it is important to highlight that the results obtained are directly related to the protocol stacks used. We used two different protocol suites, and ten different measurements were made for each suite in EdgeSec' s code running on Mobile-Hub 2. The average time taken by each operation was computed separately, and these values are shown in Table 1 and Table 2, one for each protocol suite.

Table 1 - Profiling of authentication process using BLE-RC4-HMAC-MD5

Authentication Steps (BLE, RC4, HMAC-MD5)	Time average (ms)	Standard Deviation
Send handshake hello	26.80	8.49
Read handshake response	28.10	7.18
Create Hello Message	1.50	0.50
Send Hello Message (in 3 separate BLE messages)	242.60	18.03
Receive Hello Message Response	184.20	13.91
Verify Hello Message	0.20	0.40
Total time to authenticate	**483.40**	25.64

Table 2 - Profiling of authentication process using BLE-RC4-HMAC-SHA1

Authentication Steps (BLE, RC4, HMAC-SHA1)	Time average (ms)	Standard Deviation
Send handshake hello	25.30	7.81
Read handshake response	47.60	23.35
Create Hello Message	1.60	0.49
Send Hello Message (in 4 separate BLE messages)	336.80	32.22
Receive Hello Message Response	225.00	2.37
Verify Hello Message	0.40	0.49
Total time to authenticate	**636.70**	37.98

A comparison between a standard BLE connection, and a secure connection with the authentication steps is presented in Table 3 and Table 4, highlighting the total overhead of this process.

Table 3 - Summary of authentication process using BLE-RC4-HMAC-MD5

(BLE, RC4, HMAC-MD5)	Time average (ms)	Standard Deviation
Standard BLE Connection	1075.20	40.22
Authentication Process	483.40	25.64
Total time to establish secure connection	**1558.60**	53.20

Table 4 - Summary of authentication process using BLE-RC4-HMAC-SHA1

(BLE, RC4, HMAC-SHA1)	Time average (ms)	Standard Deviation
Standard BLE Connection	1101.00	47.23
Authentication Process	636.70	37.98
Total time to establish secure connection	**1737.70**	68.07

Tables 5 and 6 show measurements for the read data operation, which takes place after two devices are already authenticated and are exchanging data securely.

Table 5 - Summary of secure read process using BLE-RC4-HMAC-MD5

(BLE, RC4, HMAC-MD5)	Time average (ms)	Standard Deviation
Standard BLE Read	86.90	4.06
Verify and decrypt message	3.50	1.75
Total time to secure read	**90.40**	5.00

Table 6 - Summary of secure read process using BLE-RC4-HMAC-SHA1

(BLE, RC4, HMAC-SHA1)	Time average (ms)	Standard Deviation
Standard BLE Read	156.10	3.91
Verify and decrypt message	5.10	2.39
Total time to secure read	**161.20**	3.89

For the BLE-RC4-HMAC-MD5 suite results from Tables 1, 3 and 5, we can see that the default time to connect with a Smart Object is around 1.075 second, and the additional overhead created by all operations of EdgeSec is around 483.4ms, resulting in an increase of 45% in connection time. Additionally, the time taken to read data through BLE is around 86.9ms, with a 3.50ms overhead for verifying signature and encrypting/decrypting a message, resulting in a total time of around 90.4ms for reading or writing data securely, an increase of less than 4%.

For BLE-RC4-HMAC-SHA1 suite results from Tables 2, 4 and 6, we have a similar pattern in results, with the values being slightly higher. 636.7ms of overhead for the authentication process, representing an increase of 57% in connection time, and 5.10ms of overhead for the read operation, representing an increase of 3%.

As mentioned before, these numbers are related to the suite of protocols used during tests. Each protocol has specific operations and mechanisms used to manipulate data, perform calculations, and execute algorithms. These operations are intrinsic to how each protocol works, and most of the time it is not possible to cut down on the resulting performance overhead. Even if we use protocols that are more efficient, and fully optimize the implementation, some parts are not in control of the framework code.

A great example of this is the BLE protocol, where we need to wrap messages into BLE Characteristics and group them into BLE Services. In EdgeSec Framework, we created a security service to group all messages related to authentication and data exchange algorithms. If we look at the Standard BLE Connection time from the performance tests, that took 1.075 second on average, more than 80% (852ms) of this was spent validating if both devices have implemented the BLE security service. This has a considerable impact on the overall performance of EdgeSec but could not be avoided when using this protocol in this specific set up.

Another example is the difference between the read operation measurements for HMAC-MD5 and HMAC-SHA1 tests. When running tests for HMAC-MD5, we got an average of 86.5ms for a standard BLE read, while for HMAC-SHA1 tests we got an average of 156.10ms for the exact same operation. This difference might be the result of various factors, such as other background tasks running on the test devices, the BLE connection latency, or the wireless signal strength. These are all factors that are not related to the authentication protocols, and we cannot directly control.

Comparing to previous works that served as foundations to EdgeSec, these measurements represent an improvement from older tests, with lower execution times for each step of the algorithm. However, the general trend was maintained, leading to the same conclusions in terms of performance, as will be discussed in section 6.2.

6.2 Results

From the results shown in section 6.1, we can see that there is a considerable overhead for connecting and authenticating two devices, while the overhead for exchanging data securely between them is negligible. During a communication session, a connection operation usually happens only once, while data exchange operation can happen multiple times. This indicates that the longer a session is, the lower the total overhead imposed by the Framework. Because of this characteristic, the requirements of the IoT system and application need to be considered when evaluating the performance impact.

For applications that do not require frequent connections and disconnections between devices, or that are not especially time sensitive (i.e., 500ms overhead during connection does not affect application behavior), EdgeSec can hugely improve security without significant performance costs. Some examples are:

- Environmental monitoring: IoT devices that are used to monitor various environmental parameters such as air quality, temperature, humidity, and more.
- Asset tracking: Tracking systems that use IoT devices to monitor the location and state of valuable assets such as vehicles, equipment, or food.
- Agricultural monitoring: IoT devices deployed in agricultural fields to monitor soil moisture levels, humidity, etc.

- Energy monitoring: Devices deployed to homes, buildings, and industrial facilities to monitor energy consumption, analyzing usage patterns, and optimizing energy distribution.

These are all use cases that would take great advantage of having security mechanisms enforced in the operation and exchange of data. And depending on how devices are deployed in the architecture, the overhead of EdgeSec would not have a significant impact.

On the other hand, applications that have a high demand for connection/disconnection operations or that are very time sensitive, might present some challenges when using EdgeSec. As we could see from the experiments, most of the time is spent performing operations that are part of underlying protocols, such as BLE. While these operations are not part of the framework core, they are required for the solution to work.

As a result, in time sensitive applications, the choice of which protocols to use may directly impact the performance overhead of the framework, determining if the impact it is acceptable or not. Examples of this type of application are:

- Autonomous vehicles: Smart vehicles packed with embedded IoT devices rely on real-time connectivity to exchange data with each other and cloud systems, making sure traffic is synchronized and capturing any external input instantaneously. Delays in connection time can have severe consequences, affecting safety, navigation, and decision-making capabilities.
- Telemedicine: IoT devices used for remote patient monitoring and other medical applications require real-time connectivity to transmit vital information, and any delay could represent a risk to patient's health.
- Industrial Automation: IoT systems in industrial automation, such as manufacturing processes or robotics, require fast and reliable connections. Delays in connection time can disrupt the synchronization of devices, leading to production line bottlenecks, quality issues, or safety risks.

6.3 Opportunities for improvements and future work

As was already discussed here, there were a few gaps in the prototype implementation that could be addressed in future work to create a more complete and coherent solution. The first one is running the authorization server on a separate machine, establishing a connection with Mobile-Hub through the internet. This would mean an additional performance cost that should be evaluated in the results analysis.

The second gap is the protocol suite negotiation during the handshake process, which would transform EdgeSec Framework into a fully dynamic and flexible solution, allowing it to be used with a larger number of devices. It could facilitate and accelerate the process of pre-registration, creating less constraints and more adaptability to new scenarios. This improvement is very important to extend

the usability of this solution and encourage more users to adopt the security framework without considerable friction or efforts.

In terms of experiments, there would be important to test the framework with a wider range of plugins and protocols, making it possible to better evaluate and understand the bottlenecks that are part of the framework implementation, and the ones that are particular to specific plugins. With more data coming from more detailed and diverse experiments, we can act to refactor and improve the Framework architecture, algorithm, and implementations, turning it into a better solution.

Additionally, based on the results analysis where we have a higher cost for frequent connections and disconnections, there is a strong opportunity for improving the performance of this solution by introducing caching mechanisms. To reduce the amount of connection operations, the same session key could be re-used across different connections.

This could be achieved by determining a certain amount of time for a session to expire. During that time, a session key remains valid. If two devices that have previously being authenticated connect again with each other, they could verify and re-utilize the same session key, avoiding a complete authentication and authorization process. This would greatly improve the performance overhead cost, and it is relatively simple to implement.

Lastly, because the framework increases the capabilities of Mobile-Hub and extends the original implementation of EdgeSec into a more complete and adaptable solution, it opens the possibility for future works. This may include:

- Improvements to the EdgeSec algorithm itself, introducing new and more modern security mechanisms.
- Use of Software Defined Networks (SDNs) to increase control and management of devices, security keys, and connections. SDNs may replace some parts of the architecture, giving more control, visualization, and fine-grained control to specific configurations of the network.
- Possibility of authenticating two Smart Objects together, using the Gateway as an intermediate node that establishes the secure connection through individual session keys.
- Extend the functionalities of each protocol interface, adding more capabilities and customization to each plugin implementation.

All these possibilities may be implemented in a way that is transparent for client applications, leveraging the concept of the framework abstraction. By just updating EdgeSec to a new version, it could bring new features and functionalities to the IoT system, without the need to change any of the client app source code.

7

Conclusion

EdgeSec Framework was proposed as an evolution of previous works that aimed at providing a security solution for IoT applications. These past works were developed and made concrete in the context of ContextNet and Mobile-Hub middlewares, creating multiple security mechanisms that ensured authentication, authorization, integrity, and confidentiality to data exchanged within a system.

However, these previous works lacked flexibility features, making it difficult to integrate new devices or protocols into the architecture. The main goal of this project was to transform these solutions into a complete security framework that could be adaptable and extensible, allowing for new security mechanisms to be easily incorporated. Additionally, some of the problems and challenges from the original implementation needed to be addressed through optimizations and refactoring so this security solution could have practical viability.

The design and development of EdgeSec Framework was successful and achieved all its main objectives. This new solution builds upon the foundations established in the original security architecture and implementations, addressing their limitations and weaknesses, and introducing new concepts and logical changes, always focusing on creating a flexible and robust security solution.

The framework consists of three main components: the framework core, an authorization server, and protocol-specific plugins and interfaces. The core executes the original EdgeSec security algorithm, with the addition of a handshake process. The authorization server defines and stores all the security information involved in the algorithm and provides access control capabilities. And the interfaces, organized into three different types, provide a common standard for new plugins to be implemented, extending the functionality and compatibility of EdgeSec.

Considering the previous works that EdgeSec Framework builds upon, and the context in which this project is inserted, we highlight the following contributions:

- The new handshake process elaborated and integrated into EdgeSec security algorithm, enhancing its robustness and adaptability by incorporating a protocol cipher negotiation. The handshake strengthens the flexibility of EdgeSec, and by extension, the flexibility of Mobile-Hub, allowing devices to dynamically decide which protocols are going to be used to communicate securely.

- Three new protocol interfaces were designed, serving as a foundation for incorporating new protocols into the solution through the implementation of plugins. By defining standardized interfaces, the framework enables seamless integration of various protocols that can be used in the algorithm, without the need of changing any code on the framework core implementation, improving flexibility and extensibility.

- The original EdgeSec implementation for Mobile-Hub 1 was completely refactored into a new highly optimized version for Mobile-Hub 2. This significantly improved the performance and efficiency of the solution,

which achieved way more acceptable results in tests and practical experiments.

Despite some challenges faced during the prototype implementation, including the incomplete handshake process, and the simulation of an authorization server as an isolated module within Mobile-Hub, the proof of concept demonstrated promising results. The overhead for connecting and authenticating devices is somewhat considerable, while the overhead for secure data exchange is negligible. This indicates that, in some cases, the framework can bring many benefits from the enhanced security, with minimal performance impact, but this will heavily depend on the type of application.

Future work should address the gaps in the prototype implementation to facilitate the use of EdgeSec Framework in real world applications. This includes running the authorization server on a separate machine to establish a connection through the internet, complete the implementation of the handshake process, expanding the compatibility with unknown edge devices, and implementing a caching mechanism for reusing session keys and reducing performance overheads. Further experiments with a wider range of plugins and protocols are also necessary to identify additional points of improvements in the framework's architecture and algorithm.

In summary, EdgeSec Framework was presented in this work as a comprehensive security solution for IoT middlewares, evolving and enhancing previous implementations to achieve better flexibility, robustness, and extensibility. By being incorporated into the many use cases of ContextNet and Mobile-Hub, EdgeSec framework can create high levels of security and make these middlewares into even more complete solutions for IoMT applications.

8

References

1 F. Dahlqvist, M. Patel, A. Rajko, J.Shulman (22 July, 2019). Growing opportunities in the Internet of Things. McKinsey & Company. Last accessed 25th June 2023: https://www.mckinsey.com/industries/private-equity-and-principal-investors/our-insights/growing-opportunities-in-the-internet-of-things

2 Internet Society. The Internet of Things (IoT): An Overview - Understanding the Issues and Challenges of a More Connected World. Accessed 25 June 2023: https://www.internetsociety.org/resources/doc/2015/iot-overview/

3 M. Endler, F. Silva, and Silva, “Past, Present and Future of the ContextNet IoMT Middleware”, In: Open Journal of Internet of Things (OJIOT), vol.4, nr. 1, pages 7-23, ISSN = 23647108, July 2018.

4 L. Talavera Rios, M. Endler, I. Vasconcelos, R. Vasconcelos, M. Cunha, F. Silva e Silva, “The Mobile-Hub Concept: Enabling Applications for the Internet of Mobile Things”, In: 12th IEEE Workshop on Managing Ubiquitous Communications and Services (MUCS 2015)

5 M. Endler, A. Silva, and R. Cruz, “An Approach for Secure Edge Computing in the Internet of Thing”s. In: 2017 1st Cyber Security in Networking Conference (CSNet), October 2017.

6 G. Cantergiani. Implementação da arquitetura de segurança EdgeSec para os Middlewares de IoMT ContextNet e Mobile-Hub. Projeto Final de Graduação, Departamento de Informática - PUC-Rio, June 2020.

7 W. Shi, J. Cao, Q. Zhang, Y. Li and L. Xu, "Edge Computing: Vision and Challenges," in IEEE Internet of Things Journal, vol. 3, no. 5, pp. 637-646, Oct. 2016, doi: 10.1109/JIOT.2016.2579198.

8 "Personal Area Network." Wikipedia, Wikimedia Foundation, 24 June 2023, en.wikipedia.org/wiki/Personal_area_network. Accessed 7 Jun. 2023.

9 “Bluetooth Technology Overview”. Bluetooth Special Interest Group. https://www.bluetooth.com/learn-about-bluetooth/tech-overview/. Accessed 8 June 2023.

10 K. Townsend. “Introduction to Bluetooth Low Energy”. Adafruit Learning. https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap. Accessed 8 June 2023.

11 K. Townsend. GATT | Introduction to Bluetooth Low Energy | Adafruit Learning. https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt. Accessed 8 June 2023

12 "Message Authentication Code." Wikipedia, Wikimedia Foundation, 5 Jul. 2023, en.wikipedia.org/wiki/Message_authentication_code. Accessed 9 June 2023.

13 "MD5." Wikipedia, Wikimedia Foundation, 5 July 2023, en.wikipedia.org/wiki/MD5. Accessed 9 June 2023.

14 "SHA-1." Wikipedia, Wikimedia Foundation, 17 Jul. 2023, en.wikipedia.org/wiki/SHA-1. Accessed 20 Jul. 2023.

15 R. Wilton, “Encryption unlocks the benefits of a thriving, trustworthy Internet”. Internet Society. https://www.internetsociety.org/resources/doc/2021/encryption-unlocks-the-benefits-of-a-thriving-trustworthy-internet/ Accessed 9 June 2023.

16 M. B. Yassein, S. Aljawarneh, E. Qawasmeh, W. Mardini and Y. Khamayseh, "Comprehensive study of symmetric key and asymmetric key encryption algorithms," 2017 International Conference on Engineering and Technology (ICET), Antalya, Turkey, 2017, pp. 1-7, doi: 10.1109/ICEngTechnol.2017.8308215.

17 "RC4." *Wikipedia*, Wikimedia Foundation, 3 Jun. 2023, en.wikipedia.org/wiki/RC4. Accessed 9 June 2023.

18 "One-Time Password." *Wikipedia*, Wikimedia Foundation, 6 Apr. 2023, en.wikipedia.org/wiki/One-time_password. Accessed 10 June 2023.

19 J. Pacheco and S. Hariri, "IoT Security Framework for Smart Cyber Infrastructures," In: 2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W), Augsburg, Germany, 2016.

20 M. Bagaa, T. Taleb, J. B. Bernabe and A. Skarmeta, "A Machine Learning Security Framework for Iot Systems," in IEEE Access, vol. 8, pp. 114066-114077, 2020.

21 S. Sridhar and S. Smys, "Intelligent security framework for iot devices cryptography based end-to-end security architecture," 2017 International Conference on Inventive Systems and Control (ICISC), Coimbatore, India, 2017.

22 R. -H. Hsu, J. Lee, T. Q. S. Quek and J. -C. Chen, "Reconfigurable Security: Edge-Computing-Based Framework for IoT," in IEEE Network, vol. 32, no. 5, pp. 92-99, September/October 2018

23 ArduinoMD5, Spaniakos | Github. https://github.com/spaniakos/ArduinoMD5 . Accessed 10 April 2023.

24 Hash-library, Stephen Brumme | Github. https://github.com/stbrumme/hash-library . Accessed 10 April 2023.

25 ESP32 Overview | Espressif Systems. https://www.espressif.com/en/products/socs/esp32/overview. Accessed 17 April 2023.

26 tiny-HMAC-c, kokke | Github. https://github.com/kokke/tiny-HMAC-c. Accessed 15 July 2023.

9
Appendix

Appendix I – Image illustrating each part of the prototype diagram.

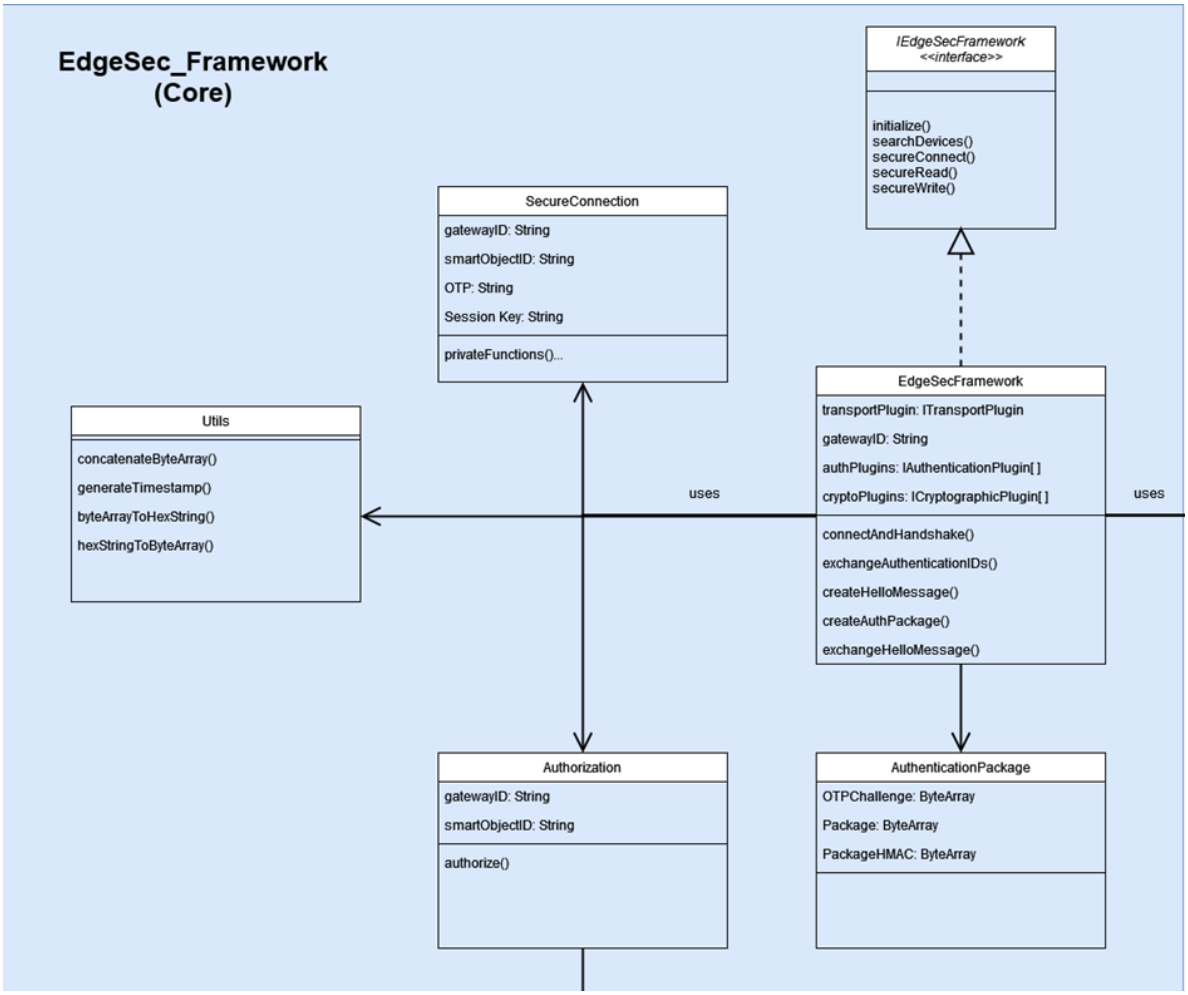

Figure 10 - Diagram of Framework Core

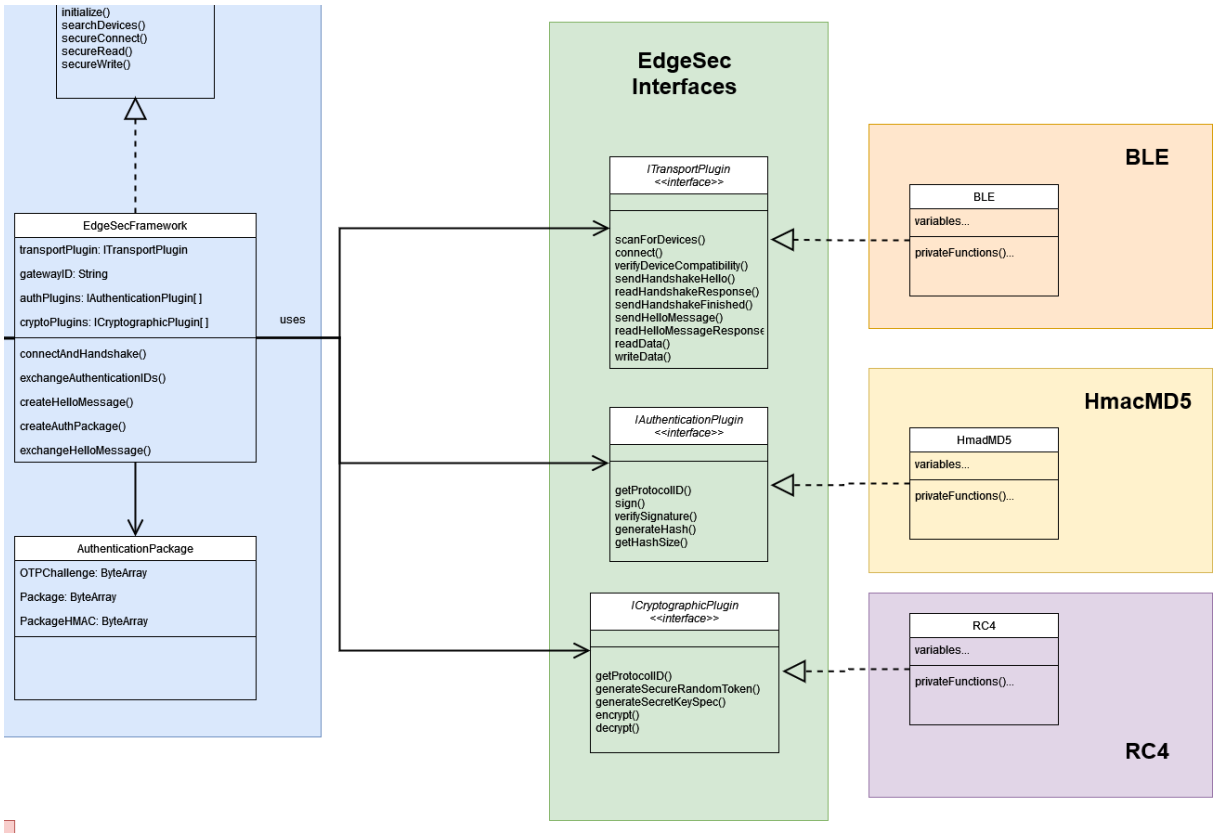

Figure 11 - Diagram of Interfaces and Plugins

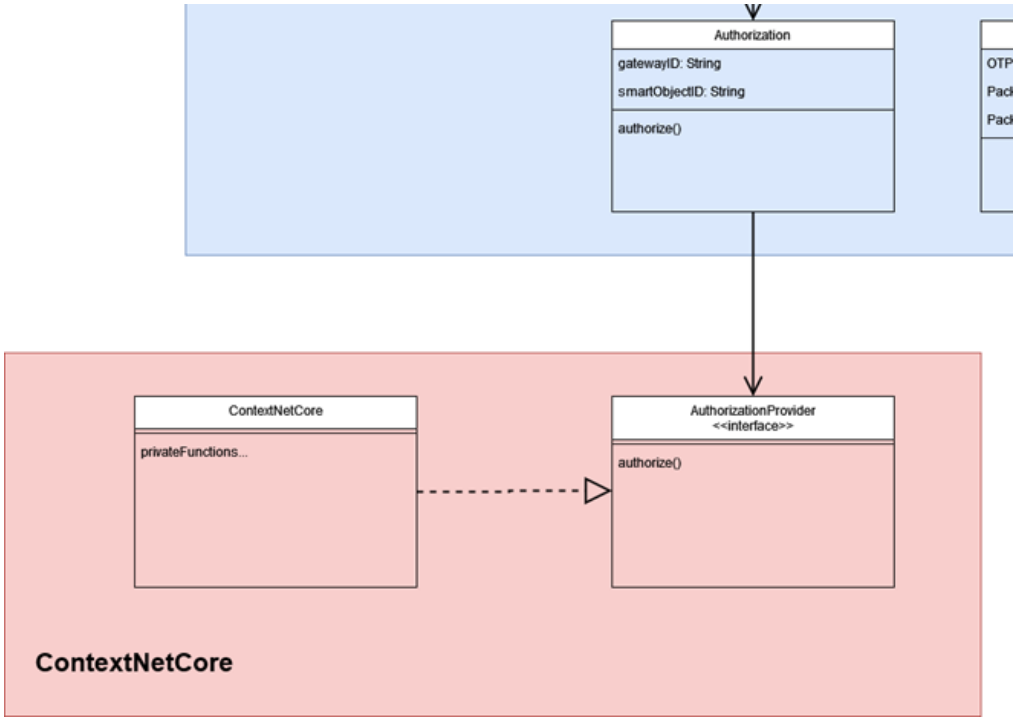

Figure 12 - Diagram of Authorization Server/ContextNetCore