A BLOCKCHAIN-BASED FIRMWARE UPDATE ARCHITECTURE FOR LONG-RANGE WIDE AREA NETWORK (LORAWAN)

by

Njabulo Sakhile Mtetwa

201409213

Dissertation in fulfillment of the requirements for the degree

Master of Science in Computer Science

Faculty of Science and Agriculture

Department of Computer Science

University of Zululand

KwaDlangezwa

RSA

Supervisor: Paul Tarwireyi

Co-supervisors: Mrs. C.N. Sibeko & Prof. A.M Abu-Mahfouz

2022

Abstract

Network security is increasingly becoming a critical and continuous issue due to technological advancements. These advancements give rise to several security threats, especially when everything is connected to the Internet. Security in IoT still requires a lot of research and it is receiving a lot of attention both in industry and academic research. IoT devices are designed for special use cases, and most are constrained in resources and lack important security features. The lack of security features enables attackers to compromise IoT devices resulting in the retrieval of sensitive information from the devices. One of the challenges in IoT is ensuring the security of firmware updates on active devices on the Internet. This is a challenge because it becomes difficult to incorporate traditional security techniques due to the limitations in memory and processing capabilities of constrained IoT devices. Thus, IoT devices remain vulnerable and open to security threats. The device manufacturers are required to release firmware updates based on exposed vulnerabilities to fix bugs and improve the functionality of the devices.

However, delivering a new version of the firmware securely to affected devices remains a challenge, especially for constrained devices and networks. This study aims to develop an architecture that utilizes Blockchain and the InterPlanentary File System (IPFS) to secure firmware transmission over a low data rate and constrained Long-Range Wide Area Network (LoRaWAN). The proposed architecture focuses on resource-constrained devices to ensure confidentiality, integrity, and authentication through symmetric algorithms by providing high availability and eliminating replay attacks. To demonstrate the usability and applicability of the architecture, a proof of concept was developed and evaluated using low-powered devices and symmetric algorithms.

The experimental results show HMAC-SHA256 as one of the symmetric algorithms utilized in the firmware update process which consumes less memory compared to the CMAC algorithm. When updating the 5 kB of firmware HMAC consumes 6.9 kB of RAM whereas CMAC consumed 7.3 kB. The memory consumption results (RAM and flash) imply that MAC algorithms are adequate in providing security on low-powered devices and are suitable for constrained low-powered devices. This conclusion is premised on the fact that the memory does not exceed the memory of the low-powered device thus, making the proposed architecture feasible for constrained and low-powered LoRaWAN devices.

Declaration

I, Mr. Njabulo Sakhile Mtetwa, hereby declare that the work presented in this dissertation is my work and that it has not previously been submitted in full or in partial fulfillment of requirements for an equivalent or higher qualification at any other recognized educational institution. All sources of information used in this work have been acknowledged.

Dedication

My dissertation is dedicated to my beloved parents who have been with me through this entire journey. I thank God for them and their endless love, patience, and constant support

Acknowledgments

I would like to express my special thanks to my supervisor Mr. P. Tarwireyi for this valuable project idea. This project provided me with the platform to pursue my interest in Blockchain technology and the Internet of Things. His constructive feedback, research guidelines, and suggestions were of great help to me.

I would also like to thank my co-supervisors Prof. A.M Abu-Mahfouz and Mrs. C.N. Sibeko. I thank Prof. A.M Abu-Mahfouz for extending the idea by introducing me to one of the amazing Internet of Things technologies, LoRaWAN. I thank Mrs. C.N. Sibeko for her academic teachings and for being with me in stressful situations. Without CSIR and the departmental financial support, this research would not have been successful.

Finally, I must express my profound gratitude to my family for their support and understanding, and for being patient with me from 2019 up to this day.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Stands for
AES	Advanced Encryption Standard
AESSKey	Advanced Encryption Standard Session Key
AMR	Automatic Meter Reading
API	Application Programming Interface
C0	Class 0
C1	Class 1
C2	Class 2
CLI	Command-Line Interface
CMAC	Cipher-Based Message Authentication Code
CPU	Central Processing Unit
CSA	Cloud Security Alliance
CTR mode	Counter Mode
CoAP	Constrained Application Protocol
DES	Data Encryption Standard
DHT	Distributed Hash Tables
DR	Data Rate
Dapp	Decentralized Application
ECDSA	Elliptic Curve Digital Signature
ETH	Ether
EU region	European region
EVM	Ethereum Virtual Machine
FOTA	Firmware over-the-air
FUS	Firmware Updates Service
HMAC	Hash-Based Message Authentication
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IETF	Internet Engineering Task Force
IPFS	InterPlanentary File System

IPVC	InterPlanetary Version Control Systems
ISM Band	Industrial Scientific Medical Radio Band
IoT	Internet of Things
K_M	Master Key
K_{MW}	Manufacturer’s Wallet Address
K_{PU}	Public Key
K_S	Session Key
LPWAN	Low-Power Wide Area Network
LoRaWAN	Long-Range Wide Area Network
MAC	Message Authentication Code
MQTT	Message Queue Telemetry Transfer
OS	Operating System
OTA	Over the Air
OTTA	Over the Air Activation
PC	Personal Computer
QoS	Quality of Service
RAM	Random Access Memory
RPC	Remote Procedure Call
RX	Receive Window
SF	Spreading Factor
SFS	Self-Certified Filesystems

List of Publications

The research has resulted in the publication of one article in an accredited academic journal and three presentations which were published as part of conference proceedings.

Accredited Journal Publication

- Mtetwa, N.S., Tarwireyi, P., Sibeko, C.N., Abu-Mahfouz, A. & Adigun, M. (2022). Blockchain-Based Security Model for LoRaWAN Firmware Updates. *Journal of Sensor and Actuator Networks*. [Online]. 11 (1). p.p. 5. Available from: http://dx.doi.org/10.3390/jsan11010005.

Conference Proceeding(s)

1. Mtetwa, N.S., Tarwireyi, P., Sibeko, N. and Abu-Mahfouz. (2020) 'OTA Firmware Updates for LoRaWAN Using Blockchain', in *International Multidisciplinary Information Technology and Engineering Conference*. Kimberley, South Africa. Available at: https://doi.org/10.1109/IMITEC50163.2020.9334108.

2. Mtetwa, N.S., Tarwireyi, P., Sibeko, N., Abu-Mahfouz A. and Adigun, M 'Secure Firmware Updates in the Internet of Things: A survey', in *Proceedings - 2019 International Multidisciplinary Information Technology and Engineering Conference, IMITEC 2019*. IEEE, pp. 1–7. Available at: https://doi.org/10.1109/IMITEC45504.2019.9015845.

3. Mtetwa, N., Tarwireyi, P. and Sibeko, N. (2019) 'Blockchain-based Architecture for Firmware Update in IoT', in *Southern Africa Telecommunication Networks and Applications Conference*, pp. 1–2. Available at: https://www.satnac.org.za/proceedings

Chapter 1: Introduction

1.1 Overview

The Internet of Things (IoT) is a network of connected sensing devices with the ability to communicate and perform tasks without human intervention (Makhdoom *et al.,* 2019). Recently, IoT has brought immense value to our lives with the ability to connect all things, people, and environments to the Internet. It has created better experiences and improved different areas of our lives which include healthcare, agriculture, smart cities, smart transport, and mobility. However, the immense benefits of IoT are not without privacy and security challenges. One of the main challenges is the software or firmware update challenge. The Open Web Application Security Project OWASP, (2018) listed software updates as one of the main challenges in IoT. The challenges with firmware updates include the lack of firmware validation on the device, lack of secure delivery, or unencrypted in transit among others.

Furthermore, firmware updates become difficult to incorporate in IoT devices in constrained networks, especially in devices that are constrained in nature or resources. Among these constrained networks are Low-Powered Wide Area Networks (LPWAN) technologies such as Long Range Wide Area (LoRaWAN), Sigfox, NarrowBand-Internet of Things (NB-IoT) with data rate ranges between 0.3 kbit/s to 50 kbits/s per channel (Gambiroza *et al.*, 2019). Besides, the constraints presented by the networks, the device constrains also contribute to the difficulty experienced in developing a secure firmware update mechanism. Several firmware update approaches and challenges for IoT devices in the context of security have been discussed in studies like Akshay *et al.,* (2019) and Mtetwa *et al.*, (2019) conducted a survey study based on the firmware updates in LPWAN and IoT generally and found that many firmware update mechanisms focus mainly on the IoT devices with enough resources. The survey study revealed that not many mechanisms focused on constrained IoT devices in constrained networks. Using the findings from the survey study, this research further investigated how firmware updates can be performed on constrained IoT devices in a constrained IoT network, particularly LoRaWAN.

1.2 IoT Challenges

The IoT merges the physical world with the digital world (Baranyi *et al.*, 2021) and has opened new opportunities for humanity. As it opens new opportunities, it has also brought challenges and security threats to companies, governments, and consumers. The security threats result from the lack of built-in security (Johnson *et al.*, 2020) due to the limitation of the central processing unit (CPU), memory, and power resources. As a result of these resource limitations, Securing the IoT device becomes a challenge due to the limitations of the resources. For example, the lack of built-in mechanisms responsible for firmware updates has been highlighted by many studies like Zandberg *et al.*, (2019); (OWASP, 2018) and is still a challenge in IoT. Firmware update mechanisms are required in times of emergency due to security breaches. Many successful attacks in the past occurred due to the vulnerabilities caused by these limitations.

One of the well-known companies, Tesla, experienced a Bluetooth attack on their Tesla Model X vehicle (Greenberg, 2020). The attack was related to a vulnerable key fog of the vehicle. The vulnerability in the key fob firmware update mechanism over Bluetooth was exploited, enabling the attacker to patch the key fob with the malicious firmware. This attack was possible due to the lack of a code signing feature in the over-the-air (OTA) firmware update mechanism of the key fob. Apart from the lack of code signing in the firmware update mechanism, other vulnerabilities were found and they all made it possible for the attacker to unlock and steal the car.

Another attack called 'Jeep Hack' (Miller and Valasek, 2015) was illustrated by a group of researchers who took advantage of vulnerabilities found in the vehicle. The firmware reverse engineering was performed which enabled the retrieval of sensitive information including, encryption algorithms, sensitive URLs, encryption, and API keys. The researchers successfully gained remote access to control a vehicle utilizing a Controller Area Network (CAN) bus that enables communication among vehicle components including brakes, steering wheel, locks, heaters, headlights, wipers, etc. The CAN messages were sent to take control of various components of the vehicle to make it accelerate or decelerate the vehicle and even veer off the road.

IoT attacks are not limited to only the automotive industry but also other areas such as healthcare. A study conducted on medical implants demonstrated the effects of firmware and communication protocol vulnerabilities on pacemakers. This attack was successful due to the lack of provision of authentication and confidentiality on the remote management channel. As a result of the vulnerabilities, it was possible to control the pacemaker's behavior, such as running the battery flat and controlling the patient's heartbeat.

These attacks have clearly shown that the security of smart devices cannot be ignored, because it can have detrimental effects not only on the affected systems but also on human lives. Thus, this shows a need for strong encryption mechanisms to ensure security during firmware updates. Table 1.1 lists different components and the vulnerabilities inherent in them (Miloslavskaya and Tolstoy, 2019).

Table 1.1 Targeted Areas of IoT

Targeted Components	**Vulnerabilities**
Device Data	Data is stored on the device unencrypted Embedded security keys information on the developed code Lack of data authentication and integrity checks both in transit and at rest. Lack of transport encryption and poorly implemented TLS/SSL enabling network traffic or data of the device to pass data in plaintext.
Device Hardware	Exposed serial ports Insecure authentication mechanism utilized in the serial ports. Open access to dump the firmware either via flash chips or JTAG firmware modification at the storage level.
Firmware Image	Insecure integrity and authentication/signature check. Outdated device components with known vulnerabilities. Hard-coded sensitive information such as passwords, and API keys, on the firmware image.
IoT Mobile Application & Web Applications	Mobile and web applications allow controlling the IoT devices, monitoring the devices, viewing analytics, controlling permissions for IoT devices, etc. Implicitly trusted by device or cloud, username enumeration, account lockout, known default credentials, weak passwords, insecure data storage, lack of transport encryption, insecure password recovery mechanism, Dumping the source code of the mobile app Client-side injection, Cross-site scripting.

Update Mechanism	Update the mechanism utilizing the unencrypted connection. The unencrypted connection enables attackers to perform malicious updates via DNS hijacking. Eavesdrop on an unsecured mechanism channel to retrieve firmware images.

There are many contributing factors to security vulnerabilities in IoT. One of them is the lack of security knowledge among developers (Votipka *et al.*, 2020). The developers of IoT devices may have limited knowledge about the security vulnerabilities of these devices. Another factor comes from the use of insecure third-party libraries and frameworks where developers utilize existing libraries and frameworks which might have potential vulnerabilities and negatively affect the developed product (Miloslavskaya and Tolstoy, 2019*)*. The security check on the code must be done before the devices or product is deployed to the Internet to eliminate any possible security breach. In addition to these causes, the development of IoT devices involves different vendors. This means that the developed IoT device comprises elements that are manufactured by different vendors. This can lead to security issues if one of the elements has vulnerabilities (Schiller *et al.*, 2022).

After the above-mentioned IoT vulnerabilities are found in IoT devices, it is then required to distribute the firmware image to the devices securely. This is done through firmware update mechanisms. Without the firmware updates mechanism, critical security vulnerabilities cannot be fixed, and IoT devices can become a permanent liability due to cyber-attacks (Zandberg *et al.*, 2019b).

The limitations of IoT devices are not the only factors contributing to IoT challenges. The protocols for handling device traffic also have challenges that restrict specific use cases. For instance, when it comes to firmware updates. One of the constrained networks with challenges when delivering firmware updates to IoT devices is LoRaWAN. LoRaWAN is the protocol responsible for handling network traffic according to Marais, Abu-Mahfouz, and Hancke, (2020), and is considered a constrained network with a low data rate including restrictions on the duty cycle and high packet loss, etc. These challenges are faced because LoRaWAN operates in the unlicensed spectrum (ISM band) and hence, cannot offer the same Quality of Service (QoS) that is offered by other networks. The restrictions mentioned above make it a challenge to apply firmware updates in LoRaWAN. For example, during the firmware update process, these limitations make it impossible for some fragments of the firmware image sent over LoRaWAN to be received by the gateway. This is due to the interference of the signal sent/packet loss, hence LoRaWAN cannot ensure successful packet delivery.

Why Decentralized and Blockchain Technology in LoRaWAN?

LoRaWAN relies on symmetric cryptography to secure the devices and to provide end-to-end encryption between the devices and LoRaWAN servers. However, with the built-in symmetric cryptography, LoRaWAN is still susceptible to some attacks (Brtnik, 2018). Recent studies have been conducted to enhance the security of LoRaWAN. One of the popular technologies that are utilized to enhance the security of IoT systems is Blockchain technology which is a decentralized peer-to-peer network (Dika and Nowostawski, 2017) that is not managed by a third party. Decentralized networks are known for their high resilience against many threats and improved scalability compared to centralized networks. The most-used firmware update approaches happen in a centralized manner, in which the IoT devices depend on a single authority for the distribution of firmware.

The central approaches make the manufacturer's server vulnerable to single-point-of-failure Witanto *et al.,* (2020) and latency issues. For example, when the manufacturer's servers are offline, there will be a delay in critical patches from being applied to IoT devices (Atzori, 2017). In a decentralized network that is not the case. A decentralized network does not allow data and processing in a single place but involves different entities that store, communicate, and process data, hence the single point of failure is eliminated. Moreover, a decentralized network like Blockchain is considered to be highly secured because it uses advanced cryptographic techniques such as hashing function and asymmetric cryptography (also known as public-key cryptography) to secure its data. The data on the Blockchain is auditable and impossible to alter or delete.

1.3 Motivation

Researchers recently conducted studies on firmware updates to come up with mechanisms that were meant to deliver updates to different types of IoT devices. Each of these recent studies either focuses on constrained networks targeting low-powered/low-end devices or on non-constrained networks targeting IoT devices with more resources. Different strategies had been developed to provide security for these devices during the update process. The client-server and decentralized Blockchain-based strategies are the two main strategies being utilized to deliver and secure firmware updates to the devices. Few studies targeted low-powered devices in constrained networks literature, particularly LoRaWAN. The existing ones only utilize the client-server approach to distribute and secure the firmware to the devices. On the other hand, Blockchain is an emerging technology known for being resilient to cyber-attack and highly secured compared to the client-server approach. However, while Blockchain is having these advantages it has not been adopted in some IoT networks. The existing Blockchain-based firmware update mechanisms focus more on medium-high-end devices in other IoT networks but, not on low-powered devices in LoRaWAN.

In addition, existing Blockchain-based techniques of delivering the firmware update focusing on another network cannot be even adopted in constrained networks, for instance, some adoptions require more resources on the devices whereas constrained networks like LoRaWAN comprise the devices that have limitations in memory and processing power. Jongboom and Stokking, (2018a) came up with some requirements or challenges that need to be addressed when delivering firmware updates to low-powered devices in LoRaWAN. Hence, most did not consider these challenges which makes them unadoptable in constrained networks. Therefore, this study proposed and implemented the Blockchain-based architecture or mechanism to deliver firmware updates to low-powered LoRaWAN.

1.4 Problem Statement

The lack of robust security solutions in IoT is an area of concern to both academia and industry. Due to their ubiquity, vulnerable IoT devices are not only a danger to the networks they connect to, but also to the humans that seek to derive utility from them (Zandberg *et al.*, 2019b). There has been a rise in the number of cases where ransomware and malware have targeted firmware vulnerabilities to cause harm, steal credentials, or even disable critical infrastructure. Research has shown that some IoT devices are even attacked within five minutes after field deployment and targeted by specific exploits within 24 hours (NetScouts, 2018). It is, therefore, evident that after the initial deployment of IoT devices, it is inevitable for vulnerabilities to be discovered (George Corser *et al.*, 2017). If not mitigated, some of the vulnerabilities will have detrimental effects. Thus, device manufacturers are expected to release new firmware versions to fix bugs and vulnerabilities to improve the device's security. The new firmware versions must be transmitted via a secure firmware update mechanism to make them available securely to the active devices on the Internet. Establishing a secure firmware update mechanism for IoT devices is a challenge. It is a challenge due to many factors such as limitations posed by IoT devices in memory and processing capabilities as well as the communication protocol with data rate limitations. Additionally, the nature of IoT comprises a massive number of geographically separated devices where some are active in areas that are difficult to reach.

This massive number of IoT devices makes it impractical and infeasible to apply manual updates to the devices active in the field because it requires remembering all devices' physical locations and then connecting each with the PC via cable to apply manual updates. Hence, there is a need for automated OTA methods of conveying firmware updates to the thousands of devices active on the Internet. The most-used OTA approaches are based on the client-server model which is a traditional model. However, this traditional approach exhibits a single point of failure therefore, there is a need for ways to convey firmware updates that use the distributed approach such as the Blockchain technology that provides high security that is resistant to conventional attacks.

1.5 Research Questions

This research aimed to answer this main research question:

How can a Blockchain-based firmware update architecture for the LoRaWAN network be designed and implemented?

From this research question, four sub-research questions emanated.

- What is the "state of the art" in LoRaWAN firmware updates?
- Why is Blockchain suitable for firmware updates in LoRaWAN?
- How can a Blockchain-based firmware update mechanism suitable for LoRaWAN be implemented?
- How can the proposed firmware update mechanism be evaluated?

1.6 Aim and Objectives

1.6.1 Research Goal

This study aimed to implement and evaluate a secure Blockchain-based firmware update mechanism that is suitable for LoRaWAN.

1.6.2 Research Objectives

The goal was broken down into the following achievable objectives:

- To establish the state of the practice of firmware updates in LoRaWAN contemporary research literature.
- To explore the suitability of Blockchain in firmware updates for LoRaWAN.
- To design and implement a Blockchain-based secure firmware update mechanism that is suitable for LoRaWAN.
- To analyze the performance of the proposed firmware update mechanism.

1.7 Research Contribution

In the recent era, the IoT network is vulnerable to many different cyber-security issues, and it keeps on growing exponentially as more of these issues are discovered. Therefore, security must be considered a major concern. In that regard, this research explored how firmware update and Blockchain-based firmware update mechanisms can be designed, implemented, and evaluated for constrained IoT devices specifically in the LoRaWAN network. To that end, we contributed in the following ways:

1.7.1 Firmware Updates in IoT and LoRaWAN

This study explored the existing approaches for delivering firmware updates to the LoRaWAN IoT network and what recent cybersecurity techniques can be utilized to implement a secure solution. Firstly, Blockchain technology was noted as one of the few approaches that are utilized for securing firmware updates in LoRaWAN, to the best of our knowledge, at the time the study was conducted there was no Blockchain security approach targeted to deliver firmware updates to a constrained LoRaWAN network. This was observed through the survey study carried out by Mtetwa et al., (2019) and which was published in a conference proceeding. Thus, this study sets a foundation for further exploration into Blockchain-based solutions for LoRaWAN.

1.7.2 Firmware Update with Blockchain Technology in LoRaWAN

The existing firmware update mechanisms for LoRaWAN are mostly done manually and some rely on the client-server model to securely distribute the firmware to constrained low-powered devices, however the client-server model exhibit the single-point-of-failure. The current LoRaWAN-based research studies demonstrate the firmware update with the use of simulation tools and to the best of our knowledge, there is no Blockchain-based study utilizing physical devices to show how constrained LoRaWAN devices can be securely updated with Blockchain technology. This study, therefore, focused on the testbed to demonstrate the possibility of firmware updates in LoRaWAN taking advantage of Blockchain technology to securely deliver firmware updates to low-powered devices. The study, therefore, demonstrated how Blockchain technology can be utilized in LoRaWAN to securely deliver firmware updates to low-powered devices by providing the design and implementation of a Blockchain-based firmware update architecture.

Here are the following implementations the study accomplished:

- an automated Blockchain-based firmware update solution in Solidity (a smart contract programming language).
- Firmware Update Service (FUS) is responsible for the entire orchestration of firmware updates. The FUS manages the entire firmware update process of low-powered devices, performs the fragmentation, and maintains the end-to-end encryption.
- The CLI script that works hand-in-hand with the FUS was implemented to help the device owners manage low-powered devices registered in Blockchain and to initiate or apply the firmware updates.
- Finally, the study implemented a decentralized application (Dapp) for manufacturers to upload firmware images and metadata to the InterPlanentary File System (IPFS) and Blockchain network respectively.

In addition, the study was evaluated to show the impact of the proposed solution and security measures taken to secure the firmware. Furthermore, it shows the overall cost involved in LoRa transmission.

1.8 Organization of this Dissertation

The remaining parts of this study are organized as follows:

Chapter 2 aims to provide the reader with background knowledge on LoRaWAN, Blockchain technology, decentralized storage, security threats or challenges available during firmware updates with the security measures. The background study is conducted to understand the design, implementation, and evaluation of our proposed architecture.

Chapter 3 this chapter reviews recent studies and approaches utilized to deliver firmware updates in IoT general and constrained networks. Apart from recent studies in firmware updates, the chapter also presents the Blockchain and LoRaWAN integration studies that aim to enhance the security of LoRaWAN. It closes by listing the benefits, limitations of each study.

Chapter 4 gives a detailed explanation of our system architecture. It starts by providing the research methodology utilized in this study. This is followed by the application scenario section, and the sections detailing the system's requirements, Blockchain smart contract design, and overall design including the security algorithm utilized.

Chapter 5 discusses how the proposed system was implemented, and what tools were utilized. It also discusses the implementation of the smart contracts, the implementation of the independently implemented FUS component responsible for the entire firmware update process, and represents the decentralized web application that helps manufacturers to distribute the firmware. The testing and validation of the smart contract operations are also provided in this chapter.

Chapter 6 provides the results and analysis of the proposed architecture and the comparison of the proposed solution against other firmware update mechanisms. The evaluation of the system's overall performance is provided by examining LoRaWAN, Blockchain, and cryptographic algorithm costs.

Chapter 7 summarises the study and presents how each research question was answered. In addition, it provides limitations and the future direction of this research.

Chapter 2: Theoretical Background

This chapter is an introduction to important concepts underpinning the study. It starts by explaining IoT networks which include the Long-Range Wide Area Network (LoRaWAN) technology. It also explains the different categories of IoT devices and the different device classes that are provided by LoRaWAN. The explanation of the Blockchain and InterPlanetary File System (IPFS) technology is followed by the security threats, and the security measures in firmware updates.

2.1 IoT Networks and LoRaWAN

This subsection explains the IoT networks and one of the constrained IoT networks specifically LoRaWAN.

2.1.1 IoT Networks

IoT is a system of devices that can communicate without human intervention (Makhdoom *et al.*, 2019). This system comprises four main components namely: sensors, connectivity, data processing, and user interface, which enable users to interact with the devices (Leverege LCC, 2018). Sensors are responsible for collecting, receiving, and exchanging data. The data sent by the sensor is carried out by a particular connectivity such as Wi-Fi, Cellular, and many others. It is then further processed to gain more insight. AWS IoT Core, AWS IoT, Analytics, Oracle IoT, Cisco IoT Cloud, and Google Cloud IoT are examples of data processing services that help to store sensor data and gain insight from it. The analyzed or processed data insight is distributed to the users via the interfaces which could be in the form of mobile applications, web applications, etc. Likewise, a user can send a message from the user interface to the sensoring IoT device. IoT devices are categorized into three main categories: low-end, middle-end, and high-end devices (Ojo *et al.*, 2018).

The low-end devices are too constrained in resources compared to other categories; their purpose is to sense, send, and sometimes receive a small amount of data without performing complex calculations. Traditional operating systems (OSs) like Linux and Windows cannot run on low-end devices, since most are low-powered or battery-powered devices and are without enough resources to accommodate these operating systems. The Internet Engineering Task Force (IETF) further subcategorizes these devices into three main subcategories: Class 0 (C0), Class 1 (C1), and Class 2 (C2). The IETF device classification is based on the device's capabilities: RAM and Flash memory are available on the device (Bormann, Ersue, and Keranen, 2014). Table 2.1 shows the subcategory of low-end devices.

Table 2.1 Classes of Low-End IoT Devices

Name	**RAM**	**Flash**
Class 0	<<10 kB	<<100 kB
Class 1	~10 kB	~100 kB
Class 2	~50 kB	~250 kB

Class 0 devices are constrained in memory and processing capabilities. They get connected to the Internet via other devices like proxies, gateways, or servers. Class 0 devices are also constrained and communicate via lightweight protocols like Constrained Application Protocol (CoAP). Class 1 can communicate with other devices on the Internet with the help of gateways. Class 2 devices are less constrained compared to other classes. They are capable of supporting protocols stack used in servers such as Hypertext Transfer Protocol (HTTP). Apart from the low-end devices, there are device types that have more resources compared to the low-end devices. These types include middle-end and high-end devices. Middle-end devices are less constrained than low-end devices and are capable of using more than one communication technology (Sivagami et al., 2021).

High-end devices have enough resources, high processing power, a lot of Random-Access Memory (RAM), and Flash memory and can run traditional OSs. Most of these devices are used as IoT gateways because of their high level of resources. The most well-known example of a high-end device is the raspberry pi. The data exchange of IoT devices is made possible by communication protocols and many protocols suitable for specific IoT devices have been developed in the past. These different methods of communication include Bluetooth, satellite, cellular, Wi-Fi, RFID, NFC, Low Power Wide-Area Networks (LPWANs), etc. Each method of communication has trade-offs between bandwidth, range, and power consumption.

These communications can be categorized into four major groups in Table 2.2:

Table 2.2 Methods of communication category.

	PAN	**LAN**	**MAN**	**WAN**
Standards	Bluetooth	IEEE 802. 11a, 802. 11b, 802.11g	802.16 MMDS, LMDS	GSM, GPRS, CDMA, 2.5-3G
Range	Short	Medium	Medium-Long	Long

Examples	NFC, IrDA, Bluetooth or Zigbee	Ethernet, fibre optics and Wi-Fi	Wi-Fi	Satellite, LPWAN (LoRa, Sigfox)

This study focused on type of WAN network particularly the LPWAN network. LPWAN has various of networks within such as LoRa, Sigfox etc. The study then focused on constrained IoT devices and constrained networks, one of the communications based on low power consumption, high range, and low bandwidth, particularly Long-Range (LoRa) and LoRa-Range Wide Area Network (LoRaWAN).

2.1.2 LoRa and LoRaWAN

Long-Range (LoRa) is a robust ISO/OSI Layer 1 wireless technique that can transmit and receive radio waves over long distances and is suitable for applications that transmit small chunks of data with low bit rates (*The Things*, 2021). LoRaWAN refers to the communication protocol and the system architecture, while LoRa refers to a physical layer. LoRaWAN system architecture is made up of different components responsible for processing LoRa packets. These components include a join-server, network server, and application server as shown in Figure 2.1. The gateway and the end devices communicate with one another via the LoRa interface.

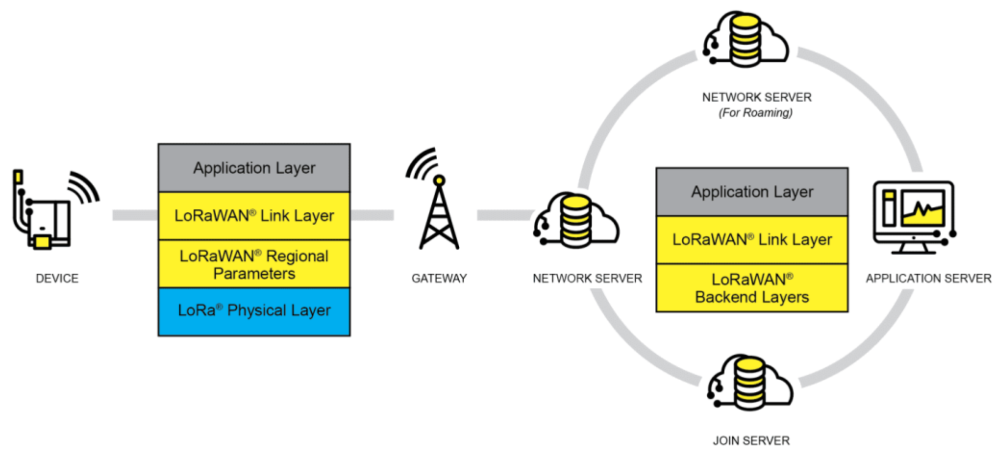


Figure 2.1 LoRaWAN Network Architecture (LoRa Alliance, 2018)

LoRaWAN Alliance specification categorizes the end devices into three classes: Class A, Class B, and Class C (Alliance, 2018). Each device class is suitable for a specific use case. The difference between the classes is based on how the devices communicate and exchange messages. When the device operates in Class A mode, communication is always initiated by the end device. The downlink message transmission is allowed after a successful uplink transmission that opens through RX windows and these windows use the channel with a low data rate. The common use cases for Class A device applications include Agriculture plant monitoring, Animal tracking, location tracking, fire detection, earthquake early detection, and more. Most of the Class A devices are often battery-powered and efficient in battery consumption since they spend a lot of time in sleep mode and are inactive in the network.

Class B device opens a receive window after sending an uplink just like the Class A device furthermore, it has an additional window that receives downlink messages. The use cases for Class B devices include temperature reporting, and utility meters monitoring of resource consumption, such as energy, water, gas, etc. A Class C mode offers the lowest latency for communication from the server to an end device it enables the device to listen and send downlink messages at any time continuously. The use cases for Class C include streetlight applications, smart utility water meters with valves, etc. With Class C mode, it is possible to schedule downlinks messages to a group of devices: this is called the multicast group. The multicast group is commonly used when delivering firmware updates to the group of end devices at the same time.

2.2 Blockchain Technology

A Blockchain is a cryptographically secure, shared, distributed ledger that stores data in an immutable manner among distributed nodes (Makhdoom et al., 2019) and is one of the implementations of Digital Ledger technologies (DLT). Blockchain maintains a list of transactions or records known as blocks. The transactions in the block are validated a verified by a special computer called miners. Miners are the ones maintaining and securing the network by working together to build trust in the network. Miners validate, verify every transaction in the network, and record it on the ledger then get rewarded with an amount of money for performing validation. To lay a good foundation and to have a good understanding of Blockchain, it is necessary to explain some important terms that are most known and used in Blockchain.:

- Gas – Refers to the unit of measurement which is the computational effort needed to execute specific operations. The different operations will result in different amounts of gas
- Gas limit – Before the transaction starts, the maximum number of units of gas that will be spent in a transaction should be available. This unit is specified by the owner of the transaction and is called the gas limit.

- Gas price – Before the transaction starts, there should be a gas price which is the amount the owner is willing to pay for each unit of gas. This is usually represented as a small fraction (gwei).
- Gas used by transaction – This is the actual amount of gas utilized by the transaction for execution. Any excess amount of gas specified during the start of the transaction will be returned to the owner.
- Gas or transaction fee – This is the product of the gas used by the transaction and the gas price which represents the actual amount of fees the transaction owner paid. It is usually presented in small fractions of the Ether (ETH), commonly referred to as gwei.

Below is an example of the successful execution of a transaction. The transaction comprises the following information:

- Gas limit (T_{GL}) and Gas price (T_{GP}) were explained above and need to be specified by the transaction owner.
- Nonce (T_N) – keep track of the transaction number for the sending account. The nonce value is incremented for each transaction made by the sender.
- To (T_{ADR}) – refers to the destination address. This can be the recipient account or the smart contract address.
- Data (T_D) – this field contains the instructions to execute a transaction in the Ethereum Virtual Machine (EVM). For example, during the smart contract deployment, T_D will have the byte code of the smart contract together with the parameters (if any) required to call the constructor function. T_D may also contain the method signature together with its parameters.
- Value (T_V) – represents the amount of Ether that is transferred between sender and recipient.
- {v, r, s} – the signature representing the transaction is represented by three variables: v, r, and s

Now, suppose Alice wants to transfer 2 amounts of Ether (T_V) to Bob. Alice will be required to specify the maximum amount of gas (e.g., 25,000) she is willing to spend for the transaction (that is T_{GL}) and specify the price (e.g., 200 gwei) for each unit of gas (that is T_{GP}). After specifying the gas limit and gas price, Alice signs the transaction with her private key. The signing mechanism is based on the Elliptic Curve Digital Signatures Algorithm (ECDSA):

$$(r, s) = ECDSA\_Signing\_Algorithm\ (Keccak256\ (T_N,\ T_{GL},\ T_{GP},\ T_{ADR},\ T_D, T_{TV}))$$

The signing algorithm takes the data produced by the sender and generates the Elliptic Curve Digital Signature Algorithm (ECDSA) signature represented by (r, s) values:

$$K_{PU} = ECDSA\_Verifing\_Algorithm\ (Keccak256\ (T_N,\ T_{GL},\ T_{GP},\ T_{ADR},\ T_D, T_{TV})), v, r, s)$$

The transaction sent by Alice will be received on the network to be validated and verified by miners using the *ECDSA_Verifing_*Algorithm. *ECDSA_Verifing_*Algorithm takes the original data produced by Alice and the signature (r, s) as inputs. The original data signature is recomputed and matched against the signature *(r, s)* produced during the signing process. The *ECDSA_Verifing_*Algorithm produces the public key and if the public key produced is Alice's key, the transaction continues. A successfully executed transaction must have used a certain amount of gas (e.g., 21,000 gas) from the initially specified gas (gas used by the transaction,) and, if the specified gas is sufficient, the transaction executes successfully, and the excess amount of gas will be sent back to Alice that is:
25,000 – 21,000 = 4,000 gas. Therefore, the total cost (gas fee) for the transaction will be computed as:

$$Gas\ fee = 21{,}000 * 200\ gwei = 4{,}200{,}000\ gwei$$

4,200,000 gwei is equivalent to 0.0042 Ether, and this is the amount the miner will receive. Therefore, Alice pays

$$Total\ Cost = 2 + 0.0042 = 2.0042\ Ether$$

The digital signatures and hashing algorithm strengthen Blockchain security. Hashing provides the immutability of Blocks in the network where each block contains the previous block's hash. Blockchain technology is known for the following unique key characteristics when compared to traditional technologies (centralized technologies):

- Decentralization: Blockchain is a peer-to-peer platform that does not rely on central authorities but is controlled and managed by multiple nodes in the network.
- Openness: Blockchain is for everyone. Anyone can become a participant and join the network to store, validate, and verify transactions.
- Auditability: All Blockchain transactions can be traced back to the Genesis Block i.e the transactions that were created at the start.

- Persistency: With Blockchain, it is impossible to update, delete, or modify the transaction once it has been created, verified, and stored in the block.

Blockchain was initially manifested in Bitcoin digital cryptocurrency (Adam and Dzang Alhassan, 2020); then, in the later stages, it was applicable in many areas such as artificial intelligence, machine learning, data sciences, augmented reality, IoT, software-defined networks, and so forth (Zarrin *et al.*, 2021). Blockchain has infrastructure that is independent of other networks and for those networks to interact with Blockchain they need to connect to interfaces. This is usually achieved through a smart contract. The smart contract is a logic that is stored and runs on the Ethereum Virtual Machine (EVM); hence it inherits the characteristics of the Blockchain. Several nodes can be utilized to access the data in the Blockchain:

- Ganache–CLI – This tool utilized in local development acts as a local Blockchain node to simulate the network. It consists of fake accounts to test and make Blockchain transactions (Truffle, 2021)
- Geth is a Blockchain client that can run on the local computer and sync with the private or public Blockchain network (Go Ethereum, 2021).
- Infura Node – this node is controlled and managed by a third party that exposes an Application Programming Interface (API) for anyone who wishes to access the node to interact with the Blockchain network (Infura, 2021).

2.3 InterPlanentary File System (IPFS)

IPFS is defined as a protocol and distributed file system that connects all computing devices with the same systems of files (Manoj Athreya *et al.*, 2021). Traditionally, the content was only accessible via the protocols like the Hypertext Transfer Protocol (HTTP), based on where is hosted using the address or location of the server. HTTP relies on the client-server model and the availability of the content managed by the central authority. If servers are down, the content becomes inaccessible on the one hand and the IPFS is based on the decentralized model. where multiple nodes hold the content in a distributed manner thus, eliminating a single point of failure. IPFS utilizes content addressing to identify and access content instead of utilizing the content’s location. IPFS is built from successful existing projects ideas or technologies which are integrated to form the distributed file system. These are the following technologies IPFS is derived from:

- Distributed Hash Tables (DHTs): A hash table is a data structure based on the key-value pair and is utilized by IPFS to find the peers that host the requested file. The file can be retrieved by querying one of the connected peers in the network since they all have the DHT which is shared and updated across all the peers in the network (IPFS, 2021a).

- Block Exchanges (Bitswap): Bitswap is a message-based protocol and a core module that handles the exchange of blocks in the network. Bitswap handles the requesting and sending of blocks to and from peers in the network. The Bitswap protocol is responsible for two main tasks: it obtains the requested blocks and sends the blocks to peers in the network (IPFS, 2021b).
- InterPlanetary Version Control Systems (IPVC): IPFS uses the version control system to provide versioning for large files and any type of content.
- Self-Certified Filesystems (SFS): IPFS is a self-certifying filesystem which means the data exchanged between peers is authenticated using a unique filename (Hackernoon, 2021). Each peer on the network comprises the node ID (uniquely identifying the peer) which is created from the node's public key. Apart from the node ID, the peer comprises the public key. During peer communication, the public keys are exchanged so that when the peer connects with others it can authenticate them. The peer authenticates another by computing the hash of the public key and comparing it against the node's ID, and, if the computed hash from the public key matches with the node ID, the node can be trusted. The combination of Blockchain and IPFS is said to be a great marriage and the future of the distributed network. Blockchain networks and IPFS are integrated because Blockchain networks have restrictions on how much data can be stored on the ledger when it comes to storage. For instance, the Bitcoin network limits the block to store not more than 1 MB of data (Vujičić, Jagodić, and Randić, 2018). Usually, a large amount of data gets stored on the IPFS, which then returns the unique identifier for data. Instead of Blockchain storing the data, it will store the returned identifier from IPFS. It should be noted, that data stored on the IPFS is tamper-proof just like in Blockchain

2.4 Firmware Updates and Cryptography

Regardless of the use cases and the class mode of the end device, having support for over-the-air (OTA) firmware updates is essential for all. Firmware updates refer to distributing binary image which contains data, calibration values, authentication secrets, and a set of instructions (firmware) that operate the hardware of the end device by telling it how to function and perform certain tasks. The purpose of the firmware updates is to fix bugs, add new functionality, and improve the security of the device. Having devices that can receive updates to their OTA is critical, especially for constrained IoT networks like LoRaWAN. In addition, some IoT devices have limited resources which makes it impossible to incorporate traditional proven cryptographic approaches to secure the firmware during the update process.

2.4.1 Security Threats and Challenges

IoT is the system of interconnected devices that changes how we live. Apart from the beauty that IoT brings. IoT devices are designed for specific use cases without paying a lot of attention to security. As devices and technology get more intelligent and more connected, the threats increase as well. Devices connected to the Internet offer a variety of security risks that affect privacy and also our health. For example, cases of pacemaker malfunction in the healthcare industry were reported (Rehman, Rehman, and Khan, 2020). A pacemaker is a body implant device that controls the patient's heartbeat. The doctors communicate with the pacemaker via an external computer called a pacemaker programmer. The pacemakers were found to be vulnerable to insecure communication protocol being utilized by the pacemaker programmer and pacemaker. The utilized protocol had no authentication scheme, and the messages were sent unencrypted. It was demonstrated that the vulnerabilities found could be used by attackers to modify the pacemaker's behavior by running its battery flat and controlling the patient's heartbeat resulting in critical condition.

A quarterly report on security breaches and cyber-attacks by (IT Governance UK, 2021) shows that firmware attacks are increasing, affecting both public sectors and businesses. The report discovered security incidents that accounted for 185,721,284 breaches. Figure 2.2 shows the affected sectors with corresponding percentages of incidents. The healthcare and health sciences sector is the sector with the highest number of security incidents. This illustrates that the breaches contribute more or bring more threats to our health.

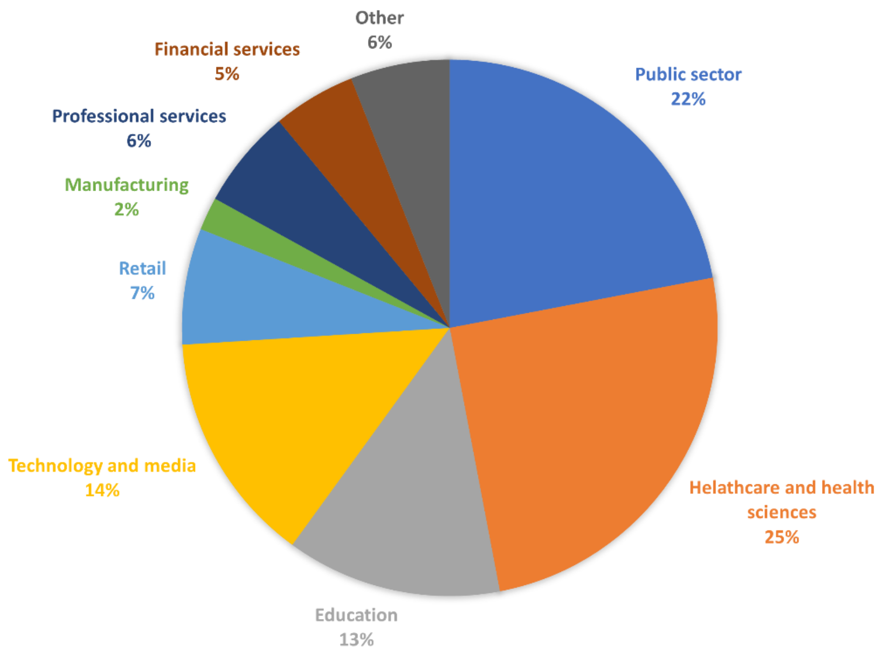


Figure 2.2 Security Breaches in Different Sectors (IT Governance UK, 2021).

These incidents do not occur only in IoT. One example is an incident that occurred to one of the world’s biggest video game publishers. Attackers penetrated the systems and retrieved the source code for FIFA 21, as well as the code for its matchmaking server (PandaSecurity, 2021). The attackers target big companies because they store customers’ important data. The attack on South Korean and Taiwan McDonald’s resulted in the compromise of customers’ data. Attackers retrieved the customer’s data including phone numbers, emails, and addresses from the system (BBC, 2021). These attacks highlighted the need for a solid mechanism to secure systems and eliminate vulnerabilities that exist within them.

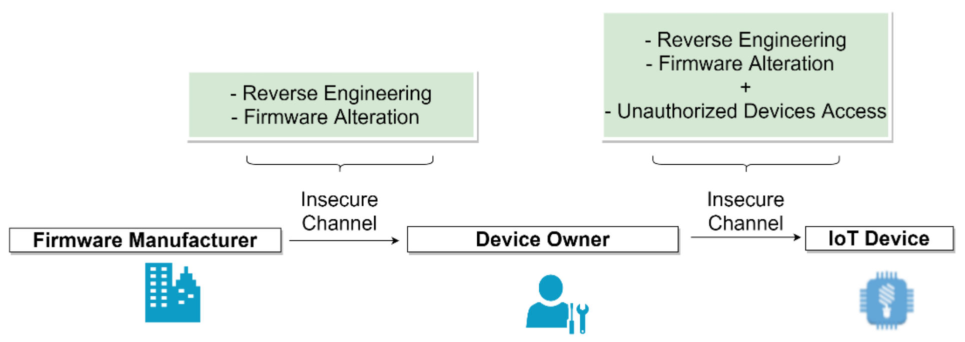


Figure 2.3 Attacks Associated with Firmware Image (Kvarda *et al.*, 2016).

When security threats occur, new firmware must be created to fix the vulnerabilities. There are threats associated with the firmware process such as reverse engineering, firmware alteration, unauthorized device access, etc (Kvarda *et al.*, 2016). The threats can occur at different levels, for instance during the transmission and after the firmware has been securely delivered to the intended device. Figure 2.3 shows the entities involved in the update process and the possible threats during the firmware transmission. The firmware manufacturer is the entity that creates the new firmware version and shares it with the device owner over the network. The communication channel between the manufacturer and the device owner can be secure or insecure.

In the insecure case, the attacker can eavesdrop on the communication and get hold of the firmware to perform firmware engineering. Extracting sensitive information from it can alter the firmware and return it for distribution. The manufacturers usually share the firmware publicly and the device owner is the one responsible for further distribution of firmware to their devices. The same threats that occur between the manufacturer and the device owner can occur during the firmware distribution of the device owner and the device. And even more threats such as aborting the update procedure, loading unauthorized firmware into the devices, and extracting sensitive information from the device such as the keys. It should be noted that the attackers have an opportunity to interrupt the update procedure in an insecure channel between the manufacturer and the device owner. The same interruption can occur on the channel between the device owner and the end device; thus, it is required to secure the firmware in transit and also at rest.

2.4.2 Security Measures, Symmetric and Asymmetric Cryptography

Bugs and vulnerabilities are inevitably going to be discovered and attacks could occur in the IoT system, therefore, security measures are needed. To fix the vulnerabilities, manufacturers need to develop a secure mechanism to deliver patches successfully to the intended device. Combining multiple security attributes such as confidentiality, integrity, and authentication also known as CIA will make a secure firmware update mechanism. Security attributes can be achieved either via symmetric or asymmetric cryptography. Symmetric cryptography requires entities to share a common secret key which is used for both encryption and decryption (TexasInstruments, 2015). Each entity must keep the shared secret key private and confidential. Any entity possessing the shared secret key can produce encrypted messages with the key and decrypt any message that is encrypted with the key. Symmetric cryptography algorithms are faster compared to asymmetric algorithms. The symmetric algorithms require the secret key upfront before the encryption and decryption can take place, in addition, the secret key must be shared securely between two entities.

Asymmetric cryptography (also known as public-key cryptography) solves the limitation of key distribution by using different keys to encrypt and decrypt messages. Messages are encrypted with a public key that is shared with everyone and decrypted with a private key that is kept secret. Due to memory and processing power limitations, asymmetric algorithms become difficult to incorporate into most IoT devices. In most cases, symmetric and asymmetric cryptography is used to provide confidentiality, integrity, and authentication.

Encryption and Decryption

Encryption and decryption ensure data privacy and enables entities with a valid key to decrypt the original message (University of Delaware, 2021). Encryption is when data is converted into an unreadable form (ciphertext) whereas decryption is the process of converting the ciphertext back to its original format. Advanced Encryption Standard (AES) and Data Encryption Standard (DES) are symmetric key algorithms examples for encryption and decryption.

Integrity

Data integrity ensures that the original message has not been modified (Sun *et al.*, 2014). In the firmware update process, this means that the firmware image generated by the manufacturer has not been modified before the device receives it. Data integrity can be achieved via different cryptographic functions such as hash functions, digital signatures, Message Integrity Code (MAC), or Message Authentication Code (MAC), etc. A hashing function takes an input (message) and produces the output that transforms data of arbitrary size into a fixed size (European Data Protection Supervisor, 2019). Given the output, it is not feasible to derive the original message, and it is also not feasible to find two different messages leading to the same hash. Apart from the hashing functions, digital signatures are also alternatives for achieving message integrity. With hashing function, any entity can produce a hash value given the message since it does not require any secret key. This means an attacker can change the message and generate the hash value and the entity receiving the message cannot detect the message alteration. Digital signatures overcome this problem by using the private key to digitally sign the message, which will be verified with the corresponding public key to determine its integrity. Digital signatures are based on asymmetric cryptography.

Symmetric cryptography can also be used to determine message integrity just like asymmetric cryptography. As mentioned earlier, MAC is one of the algorithms for achieving integrity and it is similar to digital signatures, except that it is based on symmetric cryptography and uses shared secret keys to encrypt and decrypt the message. Providing the integrity of the firmware is essential in the firmware update process. If firmware integrity is not addressed by the firmware update mechanism using cryptographic algorithms as part of security measures, the update mechanism must then implement the error correction. For instance, the CRC checksum is used to detect transmission errors during the firmware update process.

Authentication

Determining confidentiality and integrity is not sufficient because they do not prove the origin of the message. Message authentication is required to ensure the validity of the message's origin. Message authentication ensures both message integrity and authenticity of the message (Dale Liu *et al.*, 2009). It can be achieved with symmetric and asymmetric cryptography, for example through digital signatures and the MAC. In the update process, the digital signatures are used by the device manufacturer to sign the firmware image hence, the device needs to know whether is installing the authentic firmware from the manufacturer. Utilizing the MAC for authenticity will require the manufacturer and devices to have shared the same secret key in front. If the key is compromised, the attacker will be able to create illegitimate firmware with the valid MAC value which will be validated correctly by the end devices, hence it is important to keep the key secure from unintended parties.

Chapter 3: Literature Review

This chapter provides a review of relevant literature on Blockchain and LoRaWAN. It starts by looking at the existing Blockchain and LoRaWAN integration studies in Section 3.1 and is followed by Section 3.2 which explores studies that deal with firmware updates in the IoT domain based on the client-server approach. In Section 3.3, the studies that deal with firmware updates in a decentralized manner are examined. Section 3.2 and Section 3.3 describe each study while Section 3.4 provides the gaps, benefits, and limitations of these studies in Table 3.1.

3.1 LoRaWAN and Blockchain Integration

Blockchain technology has been adopted in many areas such as the IoT, machine learning, data science, argument reality, finances, and more. When it comes to LoRaWAN different studies have integrated Blockchain technology for certain purposes. The integration of LoRaWAN into a Blockchain infrastructure can be accomplished in many ways. Some researchers have studied the integration of Blockchain and LoRaWAN to enhance the security of LoRaWAN. For instance, Lin, Shen, and Miao, (2017) proposed a Blockchain-based solution to build an open, trusted, decentralized, and tamper-proof system for LoRaWAN network servers. Private organizations manage LoRaWAN networks whereas other LPWAN networks like NB-IoT are mainly managed by mobile network providers. This means LoRaWAN has to solve the issue of trust between the private network operators and the lack of network coverage. Therefore, the authors of the aforementioned article aimed to propose a conceptual architecture design for LoRaWAN network servers to solve the issue of trust of the private network operators and lack of network coverage. The authors have stated that their work is the first work that integrates Blockchain with LoRaWAN.

Durand, Gremaud, and Pasquier, (2018) proposed and built a global fully decentralized IoT network using Blockchain and LoRaWAN. The study aimed to analyze the feasibility of a fully decentralized LPWAN infrastructure and to build an architecture based on the LoRaWAN protocol. The Blockchain-built prototype focused on passive roaming techniques and benefits the crowd-sourced networks with commercial operators. The same authors (Durand, Gremaud, and Pasquier, 2018) also conducted another study that focused on taking advantage of asymmetric cryptography to provide security in LoRaWAN since it is based on symmetric cryptography. Other studies focused more on building LoRa and Blockchain-Based applications to enhance the security of a specific use case. For instance, Subodhnarayan *et al.*, (2018) proposed an Ethereum Blockchain-Based solution for the security and trust issues that are present in pollution monitoring systems. Vlachos and Hatziargyriou, (2019) presented a decentralized Blockchain-based Automatic Meter Reading (AMR) system over a LoRaWAN. The Blockchain is utilized to store the meter readings of the energy meter and ensure that the meter owner controls the data stored on the Blockchain. Ethereum Blockchain technology seems like the most dominant Blockchain utilized to provide a proof of concept and it is integrated with LoRaWAN. The Ethereum-Based proof of concept by (Ozyilmaz and Yurdakul, 2019) was also proposed to enable low-power, resource-constrained to access a Blockchain infrastructure. The study integrated Blockchain at the gateway which runs the Ethereum client to route data to the Blockchain network. A proof of concept was demonstrated using Raspberry Pi 2 connected to a Dragino LoRa/GPS Hat which acts as a LoRa node. The gateway running the Ethereum client was built using Raspberry Pi 3 combined with LoRa concentrator board iC880A.

Another study by Danish *et al.*, (2019) focused on enhancing the LoRaWAN OTTA join procedure by employing Blockchain specifically Ethereum Blockchain. The study proposed a Blockchain-based framework that adds an extra layer of security for the LoRaWAN join procedure since the join request message is not encrypted and susceptible to jamming and replay attacks. Hence, the study presents the two-factor authentication scheme for the LoRaWAN join procedure to improve authentication security and build trust among end devices and network servers. Another integration by Tan, Sun, and Li, (2021) proposed the secure architecture for LoRaWAN key management using permitted Blockchain. The study adopted the characteristics of Blockchain to come up with a scheme that avoids the single-point failure of the LoRaWAN join server and improves the performance of Over-the-Air Activation (OTAA). Additionally, the key update protocol was proposed to solve the issue of the root keys that remain unchanged once the device is created.

It was observed among the integrations provided by different studies that none was based on integrating Blockchain with LoRaWAN to deliver firmware updates in LoRaWAN securely. This was observed through the survey study that was conducted to examine firmware updates in the IoT domain (N. S. Mtetwa *et al.*, 2019). Therefore, this study proposed the Blockchain-based firmware update architecture which will run on IoT devices. The authors, Anastasiou et al., (2020) proposed a Blockchain-based framework to securely update the firmware of IoT devices using the LoRa communication protocol. The authors' study was based on the simulation tool which was implemented by Abdelfadeel et al., 2020a). The study was not clear about how the Blockchain technology was integrated with a simulation tool and what the cryptographic algorithms used in a firmware update to secure the IoT devices were and the kind of devices the proposed framework targeted.

3.2 Centralized Firmware Update Mechanisms

In recent years, privacy in the IoT domain has been a serious issue and it is being deeply investigated to provide better ways to secure IoT devices (Aqeel-ur-Rehman *et al.*, 2016). The issue of security results in various studies that try to improve different components of the IoT. Several approaches have been proposed to deal with different aspects of privacy and also numerous firmware mechanisms have been proposed to deliver firmware updates to IoT devices. Most of the proposed firmware update mechanisms can be differentiated into manual and automatic updates. Additionally, the mechanism may provide updates in a centralized or decentralized manner. This section provides an overview of those proposals that are related to the IoT domain. In particular, the focus is on the frequently used approach of client-server approach. The different tactics provided by the mechanisms for securing the firmware are also viewed.

Alexandre, (2016) proposed a solution on how to secure the firmware updates on IoT gateway devices. The proposed solution assures the firmware image's confidentiality, integrity, and authenticity and defeats the most relevant external security threats. It implemented 4 components namely: the development tool, the signing server, the update server, and the device daemon. The manufacturers used the development tool to generate the images and upload the images to the signing server. The signing server receives the firmware update images and includes keys, certificates, and configuration files in them. The IoT devices to authenticate images and the identity of the update server during TLS use these keys and certificates. The update server calculates the SHA512 hash of the firmware and signs it with its Rivest–Shamir–Adleman (RSA) private key then uploads the firmware package to the update server. It is also responsible for alerting IoT devices about new updates. The proposed solution runs a daemon that periodically sends a message to the update server to query new updates. Shortly, these entities utilize Transport Layer Security (TLS) to secure the firmware image over the channel and the firmware image is digitally signed using the private key. A checksum algorithm SHA256 is used to ensure the integrity of the firmware image. After the implementation, the solution is evaluated against network overhead and energy consumption. The authors utilized

Raspberry pi as an IoT device which is one of the high-end devices and thus the work focuses not on constrained IoT devices and IoT networks but on other unconstrained networks.

(Pycom, 2018) shows the firmware update method that targets LoRa-end devices. LoRa and Wi-Fi are communication protocols used in their update process. The method used Wi-Fi to retrieve the firmware from the servers. The reason for utilizing Wi-Fi instead of LoRa is due to LoRaWAN restrictions. These network restrictions make it hard to get the firmware image to update the devices quickly. Therefore, the mechanism switches from LoRa to Wi-Fi to access the firmware image without the delay.

Doddapaneni et al., 2017) presented a Firmware Over the Air (FOTA) procedure for IoT devices and introduced a new secure object. The work tries to improve the issues that are faced with the LwM2M protocol. Currently, the protocols like LwM2M cannot handle packet loss caused by network leakage, since the update process could be interrupted due to network leakage. Therefore, the study proposed a new secure object to save power and provide longevity on IoT devices.

Reißmann and Pape, (2017) presented an implementation of a durable and stable system for building and publishing cryptographically secure firmware updates for embedded devices based on ESP8266 microcontrollers. This includes mechanisms to build the updates from the source and automatically sign, distribute and install them on the target devices. The proposed mechanism is divided into four phases: checking for updates, reprogramming the device, calculating and verifying the cryptographic signature of the updated firmware, and reconfiguring the boot process to use the new firmware in case the update was updated successfully. The mechanism focuses on constraint devices with very low resources and uses the SHA256 and Curve25519 algorithms to secure the firmware. An approach for the secure distribution of firmware using the Message Queuing Telemetry Transport (MQTT) protocol is proposed by Borzemski (2019). The proposed approach utilizes the MQTT protocol as a communication protocol to exchange messages between the firmware broker server, the firmware server, and the gateway. The gateway communicates with IoT devices using Wi-Fi or Bluetooth. The Elliptic Curve-based Diffie-Hellman key exchange and key-hashed message authentication code are used to secure the exchanged messages and provide the authenticity of the received firmware and the integrity of the received firmware. The study conducted the security analysis to evaluate the security strength of the proposed framework.

Sahlmann et al., (2021) proposed a secure firmware update protocol for MQTT-connected devices. The protocol focused on constrained devices and ensured the authenticity of the firmware and the freshness of the firmware image. It was shown that the protocol was easy to integrate with an MQTT-based IoT network using a semantic approach. The study also provided a detailed performance analysis of the prototype implementation on an IoT device with 32 kB RAM.

Recently, there are firmware updates based on constrained LPWAN networks particularly, the LoRaWAN network. Applying firmware updates in constrained networks like LoRaWAN becomes a challenge. This is because of the network nature since they have lower data rates compared to traditional networks. The traditional network's data rate is measured in megabytes per second (Mbit/s) while the LPWANs data rate is in bits per second (bit/s). The LoRa Alliance introduced some standards to minimize the costs when performing firmware updates in LoRaWAN (Alliance et al., 2018). The standards include firmware fragmentation, clock synchronization, and multicast. Semtech (Swanson, 2020) discussed how to use these application layer packages provided by Alliance et al., (2018). Semtech demonstrated the "Fragmented Data Block Transport" and "Remote Multicast Setup" application layer packages. The discussion is based on how to provide an efficient and reliable fragmented file delivery service. The authors also explained how to use the fragmented-file delivery service which can be used to push binary firmware updates over-the-air to large groups of devices. The delivery of firmware updates to a large group of devices is achieved by utilizing the LoRaWAN protocol's unique over-the-air multicast capability in conjunction with an efficient fragment coding scheme, which significantly reduces the number of repeated transmissions required. The authors of (Jongboom and Stokking, 2018b) also made the requirements of what firmware update mechanism should consider when delivering firmware updates to low-powered devices in LPWAN. This includes how to provide the firmware update to the set of devices. The authors of Abdelfadeel et al., (2020b) used the LoRa Alliance specifications to demonstrate how firmware updates can be applied in LoRaWAN. The Firmware Update Over the Air Simulator (FUOTASim) was implemented and evaluated to demonstrate the effect of the different FUOTA parameters, however, the security of the IoT devices in this study was out of scope.

Verderame, Ruggia, and Merlo,(2021) introduced a self-protection mechanism that ensures firmware integrity through the entire production and delivery process. The authors of the work proposed the self-protection mechanism because most of the existing mechanisms lack proper integrity verification, leaving firmware exposed to repackaging attacks. The mechanism eliminates the use of signing certificates therefore, the security is provided without requiring external trust anchors or verification processes. Techniques, (2021) investigated adaptive data rate (ADR) techniques in an application that monitors cattle utilizing low-powered devices. The low-powered devices transmit the cattle's location and health using LoRa. The study also focused on the issues related to firmware updates such as speed, and reliability concerns with security when updating the firmware to both mobile and low-powered devices. The study state that the firmware update could be interrupted when the cattle are moving out of the transmission range, or the device battery may not be adequate to finish the update process. The study addressed this issue by proposing a secure and reliable firmware update process using ADR techniques that is suitable for mobile or low-powered LoRa device. conducted the experiments via simulation that focuses on LoRaWAN to examine the impact of multiple gateways during the firmware update process. The impact of multiple gateways was investigated since the single gateway cannot optimize the firmware update over-the-air (FUOTA) mechanism. The authors Charilaou *et al.*, (2021) extended the FUOTAsim simulation tool to support multiple gateways. The results of this study have shown that several gateways can eliminate the trade-offs that appeared using a single gateway. The contributions of the authors include the investigation of the impact of multiple gateways by analyzing the network's behavior during the firmware update process by varying the firmware size and network parameters. The authors also investigate the minimum set of gateways that can be used to provide full coverage with the greatest performance during the firmware update procedure. and the final contribution of the study provides insights between the firmware size and the number of gateways. The study focused only on the IoT gateway not particularly on the end IoT devices.

3.3 Decentralized Firmware Update Mechanisms

One of the popular ways of transmitting firmware to IoT devices is to transmit it in a decentralized and distributed manner. Distributing the firmware in this manner has more benefits compared to the client-server-based distribution (Makhdoom et al., 2019). In this section, the different approaches proposed to convey the firmware in a decentralized manner are examined critically with a focus on the decentralized firmware updates utilizing Blockchain technology. Lee and Lee, (2017) proposed a Blockchain-based scheme that focuses to secure embedded devices in the IoT environment. The proposed scheme relies on Blockchain technology to verify the firmware version and validate the firmware's correctness. The IoT device acts as a Blockchain node on the network, meaning it is required to store the Blockchain ledger. This becomes a challenge since many IoT devices have limited resources such as energy, computation, and storage capacity. Hence, the mechanism might be difficult to incorporate in IoT-constrained devices in the IoT environment. Yohan and Lo, (2019) proposed framework that focuses on providing secure verification of the firmware. The proposed firmware update framework consists of four processes: the creation of firmware update contract, the creation of firmware replication contract, the direct firmware update mechanism, and the indirect firmware update mechanism. The framework only ensures the correctness of the firmware version and only provides security integrity.

Mtetwa, Tarwireyi, and Adigun, (2019) proposed the Blockchain-based where Ethereum Blockchain and IPFS were used to store the firmware metadata and firmware image respectively to achieve high availability. The proposed mechanism ensures the integrity of the firmware file and targets the devices with enough resources to carry out cryptographic operations that are the high-end devices. The raspberry pi was used to test the proposed mechanism.

Witanto et al., (2020) proposed two techniques that deliver firmware updates. One of the techniques is a direct firmware update that is based on the client-server model. This technique enables IoT devices to download the update from the manufacturer’s server via the IoT gateways which then share the downloaded firmware updates from the manufacturer’s server. IoT gateways perform the integrity check and validity of the update to the Blockchain network. The second technique is a distributed peer-to-peer technique. This technique uses the Blockchain contract to check the firmware updates. The proposed techniques work well with IoT devices that have sufficient resources but not for constrained IoT devices with limited storage. The reason for this is that the IoT devices need to hold the firmware and share the available newest update through IoT gateways.

Fukuda and Omote, (2021) proposed a firmware distribution method that reduces gas costs, using a contract and access control. The proposed scheme was evaluated, and the results show that the proposed scheme successfully lowers the gas cost required for firmware updates. Some studies are based on a constrained network where Blockchain is used for securing the data. In one of our survey research studies, Mtetwa *et al.*, (2019), several IoT firmware schemes were examined. The examined studies were based in the IoT environment focusing on IoT devices with limited resources and the ones with adequate resources to handle firmware updates. The study revealed that there was a need for a Blockchain-based firmware update mechanism that targets IoT devices with limited resources.

After our survey study has been conducted, Anastasiou et al., (2020) proposed a Blockchain-based framework to securely update the firmware of the IoT devices using the LoRa communication protocol. The proposed framework was based on the simulation tool which was implemented by Abdelfadeel et al., (2020a). However, the work is not clear on the cryptographic algorithms used to secure the end device and what kind of devices the proposed framework targets. Another Blockchain-based study by (Tsaur, Chang and Chen, 2022) where a secure and efficient protection mechanism that is based on blockchain technology was proposed. The study prompts to improve traditional update methods security and also reduce the need for storage space. The proposed solution aims at integrity, device anonymity, security, and system security. In addition, the study compares the solution with the existing one. Sanchez-Gomez *et al.*, (2021) presented a solution that provides firmware update distribution and trust monitoring. The presented solution leverages LoRaWAN, Low-Overhead EAP over CoAP (LO-CoAP-EAP), a novel lightweight bootstrapping protocol, a wide-spread long-range communication technology, IPv6 header compression and fragmentation mechanism, The Object Security for Constrained RESTful (OSCORE), end-to-end application-layer protection, decentralized IPFS network, and hyper ledger as a distributed ledger technology for secure validation of the distributed information.

3.4 Benefits, Limitations, And Summary of Firmware Mechanisms

The previous sections review the related work that focuses on firmware updates in IoT. These works provide certain benefits and limitations in the IoT context. Hence, this section provides clear limitations and benefits for each study as shown in Table 3.1.

Table 3.1 Contributions and Limitations of Server-Based Firmware Approach

References	**Benefit(s)**	**Limitation(s)**
	Centralized-based	
(Alexandre, 2016)	Covers basic security threats include confidentiality, integrity, and authenticity. Proposed, implemented, and evaluated the proposed mechanism.	May not be suitable for too constrained IoT devices, since it uses an RSA signature which may not be incorporable to some IoT devices.
(Pycom, 2018)	Accommodates constrained devices specifically low-	The approach requires that the end device must be

References	Benefit(s)	Limitation(s)
	powered LoRa devices. Uses both LoRa and Wi-Fi to transmit firmware.	equipped with Wi-Fi which other devices may not have. Moreover, the mechanism is good for devices that are not battery-powered, otherwise using traditional technologies may consume the battery of the end device.
(Doddapaneni *et al.*, 2017)	Demonstrated firmware update procedure that can handle loss packets in the lossy network.	Only proposes the FOSE and no implementation, or analysis of the proposed work.
(Reißmann and Pape, 2017)	This paper focused more on the implementation of firmware updates based on ESP8266 microcontrollers. Implemented and evaluated the solution.	No evaluation and performance analysis. Incompatible with constrained networks.
(Lo and Hsu, 2019)	Discussed the security analysis of ECSH key exchange, man-in-the-middle, and replay attack	The study is on the client-server model and uses subscribe publish model. Each manufacturer manages its broker's bad patch server. This model may scale well with a large amount of IoT devices. Only proposed but have not implemented the scheme.
(Abdelfadeel *et al.*, 2020b)	Demonstrated how the LoRa firmware standards or specifications can be used to provide the firmware update	The work focuses more on how the firmware update can be done on a large scale in

References	Benefit(s)	Limitation(s)
	to a large number of devices using the simulation tool.	LoRaWAN but does not cover the security part of it.
(Sahlmann *et al.*, 2021)	Discussed existing updates for constrained devices that use MQTT protocol to transmit firmware.	Based on the client-server model may scale well with a large amount of IoT devices.
(Techniques, 2021)	Presents a firmware update methodology for both mobile and low-powered devices.	The methodology demonstrates the update mechanism using only a single device. In other words, the mechanism must be able to serve firmware updates to the set of devices in the IoT network.
(Charilaou *et al.*, 2021)	Provides support for the utilization of multiple gateways instead of a single gateway during firmware updates.	Some security properties can be improved for example providing the end-to-end encryption between the Firmware Update Server and the LoRaWAN servers and providing data confidentiality between the Firmware Update Server and the end devices.
	Blockchain-based	
(Lee and Lee, 2017)	Discussed the Blockchain-based scheme that provides high availability, integrity, and authentication in-depth.	The proposed mechanism might be difficult to incorporate in constrained IoT devices because of limited resources. The scheme has not been implemented and evaluated.

References	Benefit(s)	Limitation(s)
(Yohan and Lo, 2019)	Discussed feature comparison between the proposed firmware update framework and existing frameworks	The limited literature on firmware updates. The mechanism only ensures the integrity of the firmware image. No implementation and evaluation only proposed the mechanism.
(Mtetwa *et al.*, 2019)	The study focuses in-depth on how the software update may take place with Blockchain utilized to secure the entire process.	The proposed mechanism targets IoT devices but is not suitable for devices that are too constrained in resources.
(Witanto *et al.*, 2020)	The work provides two ways of updating IoT devices, the client-server and the distributed approach. Discuss the implementation and analysis of these two techniques.	The proposed techniques are good for IoT devices with sufficient resources but not for constrained devices with limited storage.
(Anastasiou *et al.*, 2020)	Discussed how Low-powered devices can be updated based on the simulation tool.	The study performs firmware updates utilizing Blockchain but it is not clear how the Blockchain was implemented in the simulation tool developed by (Abdelfadeel *et al.*, 2020b). The mechanism claims to provide authenticity and integrity but does not specify which algorithms are utilized to achieve such. Moreover,

References	Benefit(s)	Limitation(s)
		no security analysis was performed.
(Fukuda and Omote, 2021)	Explored previous updates method considering incentivize and provides the comparison of the proposed scheme with the previous ones based on the gas cost.	Not clear which encryption and hashing algorithm is used for confidentiality and integrity. Not compatible with the constrained network.
(Tsaur, Chang and Chen, 2022)	Focuses on security goals such as Malicious code resistance and Distributed denial-of-service (DDoS) resistance.	Fewer comparisons against available existing Blockchain solutions, only three studies were compared. The fundamental security goal achieved is firmware integrity only with no confidentiality and authentication.
(Sanchez-gomez *et al.*, 2021)	Enables trust-worthy management of large heterogeneous IoT networks for firmware update distribution.	The implementation and execution details of the platform are not provided as yet, it is part of future work. This includes the test validation results of the details of the operation together with the performance analysis and scalability testing.

The authors proposed different mechanisms that provide certain security properties in the firmware update process. These properties include availability, confidentiality, integrity, authentication, and data freshness. Each mechanism may target a certain group of devices, including low-end, middle-end, and high-end. Some of these mechanisms target constrained networks and constrained devices while others do not. Moreover, some of the mechanisms were evaluated and the security analysis was provided while some were proposed without performing any evaluations on how the mechanism behaves. Table 3.2 gives a summary of all the mechanisms and shows which properties were achieved, what type of approach was taken to provide firmware update, which type of IoT network it targets e.g., constrained network, and finally if the mechanism was evaluated. These mechanisms are categorized based on the client-server and Blockchain-based or decentralized models.

Table 3.2 Comparison between proposed approaches

Authors	Low/Middle-End Device	High-End Device	Constrained-network	Availability	Confidentiality	Integrity	Authentication	Data Freshness	Performance Evaluation
Centralized-Based									
(Alexandre, 2016)		✓			✓	✓	✓		✓
(Pycom, 2018)	✓		✓			✓			
(Doddapaneni *et al.*, 2017)	✓		✓			✓			
(Reißmann and Pape, 2017)	✓					✓	✓		
(Lo and Hsu, 2019)			✓			✓	✓		
(Abdelfadeel *et al.*, 2020b)	✓		✓						✓
(Sahlmann *et al.*, 2021)	✓				✓	✓	✓	✓	✓
(Verderame *et al.*, 2021)	✓				✓	✓			✓
(Techniques, 2021)	✓		✓		✓	✓	✓		✓
(Charilaou *et al.*, 2021)	✓		✓			✓	✓		✓
Blockchain-Based									
(Lee and Lee, 2017)		✓		✓	✓	✓	✓		
(Yohan and Lo, 2019)		✓				✓			
(Mtetwa *et al.*, 2019)		✓		✓		✓			
(Witanto *et al.*, 2020)		✓		✓		✓	✓		✓
(Anastasiou *et al.*, 2020)				✓		✓	✓		✓
(Fukuda and Omote, 2021)		✓		✓	✓	✓	✓		✓
(Sanchez-gomez *et al.*, 2021)	✓		✓	✓	✓	✓	✓		
(Tsaur, Chang and Chen, 2022)		✓	✓						

Table 3.2 shows that a lot of the update mechanisms ensure the confidentiality, integrity, and authentication properties and some add extra properties such as availability, and data freshness. It is observed that most of the client-server-based mechanisms target low-end devices and middle-end devices and none of the Blockchain-based mechanisms targeted low-end devices. Note that the low-end and middle-end devices are also called low-powered devices because a low-powered device may belong either to the low-end or middle-end class.

From Section 3.1, it was stated that there was no integration focused on delivering firmware updates to LoRaWAN using Blockchain technology when this study was conducted. This was observed through our survey study by Mtetwa et al., (2019) that was conducted and published as a journal article. However, after the survey study, Anastasiou *et al.*, (2020) conducted a Blockchain-based study that utilizes a simulation tool developed by Abdelfadeel *et al.*, (2020b). The study used the simulation tool and was not clear how Blockchain was integrated into the simulation tool and does not show any real-world implementation of Blockchain being integrated with LoRaWAN. Apart from this study, most of the tactics used in the update process of the mechanisms make the mechanism incompatible in constrained networks. Perhaps they do not aim to utilize the Blockchain to provide security in the constrained networks. This suggests the need to develop a real-world mechanism that utilizes Blockchain technology to secure low-powered devices. Therefore, this research proposed a Blockchain-based mechanism that promises to deliver firmware updates for low-powered devices in LoRaWAN. The Blockchain in LoRaWAN is used to make the distribution of the firmware decentralized and enhance security during the firmware updates.

Chapter 4: Research Design and Methodology

The literature review presented in Chapter 3 showed the need to develop a firmware update architecture that focuses on low-powered devices in the Long-Range Wide Area Network (LoRaWAN). This chapter presents an architecture based on Blockchain technology to deliver firmware updates to low-powered IoT devices in the LoRaWAN network. Before the architecture is presented the methodology is presented after which the application scenario is provided for clarity on where the architecture can be utilized.

4.1 Research Methods

Research in the discipline of Computer Science (CS) and Information Systems (IS) is generally carried out via the utility of one or more research methods. The most-used study methods for CS and IS include simulation, design prototypes, design science, surveys, and experimental methods.

The Simulation Method

The simulation methods are widely used in the CS field as they offer the possibility to study systems outside of the experimental field or the system under development or construction. This can involve complex events that cannot be performed in a laboratory. Areas that often involve simulation include astronomy, physics, economics, and specialties such as the study of artificial life, virtual reality, or nonlinear systems.

The Design Prototyping

The prototyping method enables the layout of a working "prototype". Prototyping is frequently used to predetermine a large portion of resource deployment in development and influences the success of design projects (Dooley, 2002).

The Experimentation Method

The experimentation method refers to the task of conducting real-life experiments. Experiments are often used to test truth and theories. The experimental research method is widely used in proof automation, natural languages, performance, and behavioral analysis (Fatjon Muca, 2014).

The Design Science

Design science research is a qualitative research approach in which the object of study is the design process. Design science generates knowledge about the method used to design an artifact (Carstensen and Bernhard, 2019). The Design Science research methodology incorporates, practices, procedures, and principles necessary to carry out the research with three main objectives: it is consistent with past literature, it offers a nominal procedure model for undertaking research, and it provides a mental model to present and evaluate the research.

4.2 Research Selection

Since this study introduces an artifact, prototyping and experimentation were the most appropriate methods to use. This study required that experiments be conducted to determine how well the proposed architecture behaves in low-powered devices and LoRaWAN-constrained networks. In this research, the aim was to design, implement and evaluate a secure Blockchain-based firmware update mechanism that is suitable for LoRaWAN. While conducting research, it is important to ensure that the selected methods align with the objective of the study. A survey method was first used to establish the literature on an existing firmware update in LoRaWAN examining different approaches to be utilized to deliver firmware updates.

In addition, the method also examines the state of the art in Blockchain and LoRaWAN integration. The problem analysis and the literature review were conducted to fully understand the need to develop a secure Blockchain-based solution for LoRaWAN and to identify the gaps that need to be addressed in firmware updates in LoRaWAN. Subsequently, this research adopted two research methods that align with the research objectives and provide answers to the research questions described in Chapter 1. The first method is the prototyping method. This method was the most appropriate for the implementation of the Blockchain-based firmware update solution for LoRaWAN. The second method selected was the experimental method. This method was adopted because it provides an important paradigm for conducting applicable yet rigorous research. In the following Subsections, the use of the selected methods is described.

4.2.1 Prototyping

Based on the design criteria gathered from the literature, a prototype that adheres to low-powered devices specification was designed and then implemented. The prototype was implemented to meet the requirements gathered in the literature survey.

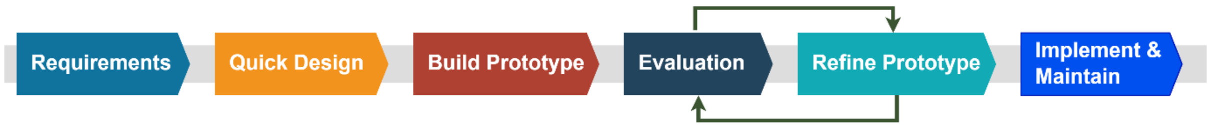


Figure 4.1 Prototyping stages.

LoRaWAN is one of the Low-Powered Wide Area Network (LPWAN) technology that was utilized as a protocol. Blockchain was integrated into LoRaWAN, to enforce the security in the firmware update process. For every requirement, the preliminary design was conducted. In the design stage, a simple design of the firmware update mechanism is created. This preliminary design helps in developing the prototype. The next stage represents an actual prototype which is based on the preliminary design from the previous stage. This stage outputs a small working model of the required firmware architecture. Once a small working prototype is produced the next stage deals with the evaluation of the prototype to help discover the strength and weaknesses of the working architecture. A working architecture prototype is refined until all the requirements gathered in the literature are met. Once the final architecture is developed based on the final prototype, it is thoroughly tested and deployed to both Blockchain and LoRaWAN networks.

4.2.2 Experimentation

This research method aimed to examine how well the proposed solution performs. This includes examining whether the architecture suits the constrained low-powered devices, by examining energy consumption, memory consumption, cost of proposed Blockchain smart contract operations, and the time the solution takes to update the devices. Hence, the aforementioned properties are the metrics used in the proposed architecture evaluation. The experiment utilized physical low-powered devices. These are LoPy devices from Pycom, LoRa gateway, and personal computers. Table 4.1 shows the experimental devices respectively.

Table 4.1 Devices Specification.

Devices	RAM	Storage	CPU	Model	Devices
Pycom Lopy	4 MB	8 MB	Espressif ESP32 chipset	LoPy4	2
LoRa Gateway	1GB LPDDR2 SDRAM	4GB SD Card	Cortex-A53 64-bit SoC @ 1.4GHz	Raspberry Pi 3 Model B+	1
PC	8 GB	SSD	4th gen Intel® Core™ i5	Lenovo T440p	1

The LoPy devices served as LoRa end devices, the gateway was built using the RAK831 module combined with the Raspberry Pi 3 B model. The personal computer (PC) was utilized to run the LoRaWAN network server which handles LoRa packets.

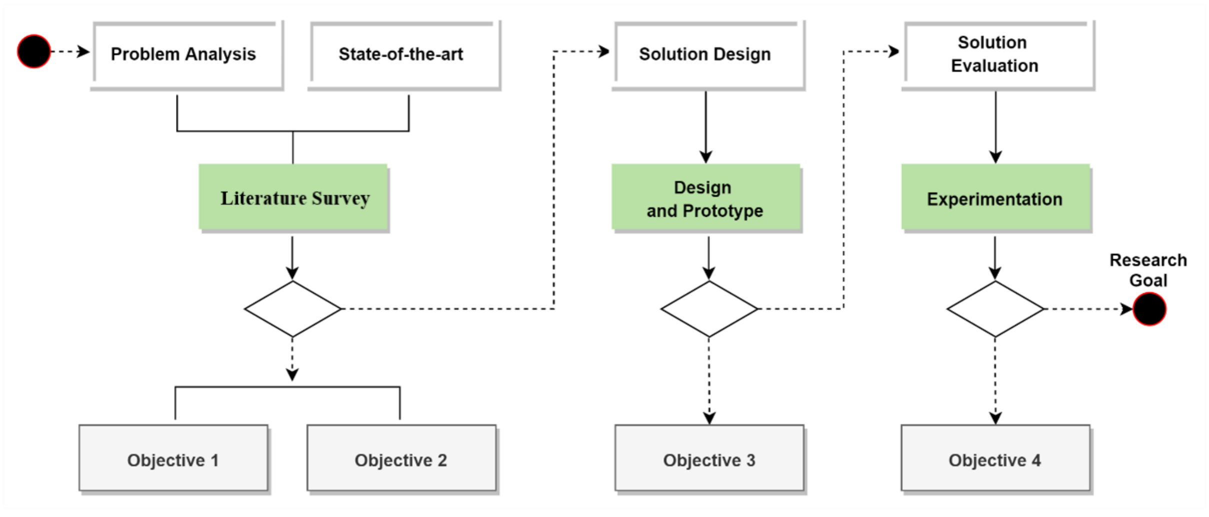


Figure 4.2 Overview of the Research Methodology

The overview of the research methodology is illustrated in Figure 4.2. The literature survey method aimed to answer the first two research questions and to achieve the first two objectives. The prototype method answers the third research question to achieve the third objective and the experimentation method answers the fourth research question and achieves the fourth objective. All these methods culminate in achieving the research goal as illustrated in Figure 4.2.

4.3 Requirements and Assumptions

This section presents the requirements (REQs) of the proposed work with the assumptions as well as the reasons for the given assumptions. Some of the requirements presented are based on the studies covered in Chapter 3: They are also influenced by the firmware updates recommendations made in publications such as Cloud Security Alliance (CSA), NIST Special Publication (Regenscheid, 2018), and other recommendations on performing updates in LPWAN networks (Jongboom and Stokking, 2018b).

4.3.1 Security Requirements of the System

The proposed work has seven requirements as explained in Table 4.2.

Table 4.2 System requirements

Requirement Number	**Description**
REQ1	**Push Updates –** The system should enable administrators or device owners to schedule firmware updates to their devices to avoid network saturation and limit unintended downtime.
REQ2	**Manage Updates** – One component or entity must manage updates of multiple IoT devices.
REQ3	**Over-The-Air Updates –** The firmware update mechanism must adopt the over-the-air strategy, and be adapted to the network bandwidth constraints.
REQ4	Updates mechanism should provide end-to-end security, authentication, and integrity which must be protected • **Authentication** – The architecture must be able to identify the origin of the firmware image or any sensitive information. For example, the architecture must be able to authenticate if the firmware image comes from a legitimate manufacturer before it gets installed on the end device. • **Integrity** – The data must be protected from unauthorized changes to ensure reliability and correctness. Integrity assures the accuracy and completeness of any sensitive firmware information. The sensitive information should be protected both at rest and in transit between systems. Therefore, the architecture should be able to check if the sensitive information, has not been modified both at rest and in transit. • **Confidentiality** – Any sensitive data must be protected from unauthorized viewing and other access. The firmware update process consists of the firmware updating sensitive information transmitted between entities of the system. Hence, the information must be viewed by the authorized entity.
REQ5	**Availability** – The end device must not rely on the central repository to receive firmware updates, the device must be updated regardless of whether the manufacturer's servers are offline or not. This means that there must be no single point of failure during the updates
REQ6	**Replay Attack** – The architecture has to consider that no old messages are allowed during the update process. The attacker can try to eavesdrop on the exchanged messages during the update and later try to replay the same messages to disrupt the updates. Thus, the system must be tactful and resilient against such actions.
REQ7	**Low-power consumption** – The architecture must accommodate resource-constrained and low-powered (battery-powered) devices in the LoRaWAN network. In addition, it must be able to accommodate devices

	without any WiFi interface or devices that cannot connect directly to the Internet.

4.3.2 Research Assumptions

The proposed architecture has three assumptions listed as follows.

- Encryption and decryption keys are on the secure hardware module. This means that the security keys reside in a place that cannot be manipulated or retrieved by bad entities like attackers.
- The firmware image is stored in public storage where anyone can access it. Usually, the firmware is stored on the manufacturer's website and the device owner can download the firmware from that public repository of the manufacturer.
- The firmware updates are assumed to be applied on battery-powered or low-powered constrained devices with low processing capabilities and memory limitations.

4.4 Proposed Architecture

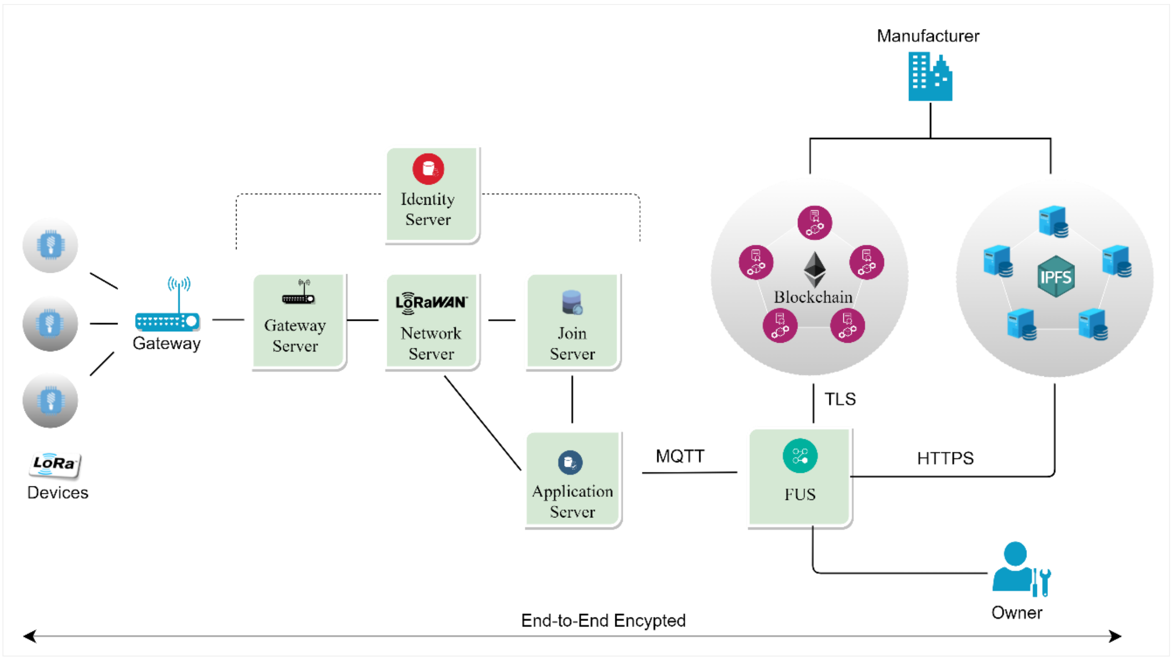


Figure 4.3 System Architecture

The system architecture comprises eight main components as illustrated in Figure 4.3. These are the LoRa end devices (low-powered devices), LoRa gateway, LoRaWAN servers, firmware update service (FUS), Blockchain, IPFS, device manufacturer, and the device owner. This section explains each component, the role it plays in the proposed architecture, and the part of the requirements it fulfills.

4.4.1 LoRaWAN System Components

Low-powered Device

This is the entity equipped with LoRa and needs to be updated over time. A low-powered device communicates with other system entities via the LoRa gateway.

The low-powered device chosen is a LoRa end device. The Low-powered device serves as the constrained IoT device that provides the LoRa interface; this enables us to send firmware updates to the end device without any internet connection. Utilizing this component helps to meet REQ7 where the low-powered device is required for demonstrating the proposed architecture.

LoRa Gateway

LoRa gateway connects Low-powered-devices to the outside world of LoRa. It receives and transmits data from multiple end devices and sends data to the network server for further manipulation.

The main purpose of this component is to facilitate communication between the servers and IoT devices without the need for low-powered devices to require any internet connection to communicate with the servers. In addition, a gateway helps to provide long-range communication for low-powered devices since most are deployed in areas with no electricity and they are required to communicate with the internet. This component helps to meet the REQ3 (helps low-powered devices to receive over-the-air updates) and REQ7.

Manufacturer

The device manufacturer is the entity responsible for the creation of new firmware and publishing the newly created firmware to the IPFS and Blockchain network. The manufacturer is one of the main entities in firmware updates. It has no more role than uploading the newly created firmware to the Blockchain network.

Device Owner

The device owner owns the low-powered device as well as the FUS which is connected both to IPFS and Blockchain network. The device owner is responsible for managing the Low-powered devices and can initiate the update process to update the owned devices. Hence, the owner interacts with FUS to manage and initiate the update.

Since the IoT network consists of a massive number of devices, and they need to be maintained by the device owner, the device owner entity helps to meet the REQ1 and uses the command line tool developed in this study to maintain the devices.

LoRaWAN Servers

LoRaWAN servers handle packets sent from the LoRa gateway and the FUS. They all form part of the LoRaWAN server for processing LoRaWAN packets. These servers are explained below:

Gateway Servers

The gateway server is responsible for maintaining connections with gateways that support the UDP, Message Queue Telemetry Transfer (MQTT), Google Remote Procedure Call (gRPC), and Basic Station protocols. It forwards uplink messages to Network Servers and schedules downlink messages to the end devices via a LoRa gateway

It is also responsible for maintaining connections with gateways that support the UDP, Message Queue Telemetry Transfer (MQTT), Google Remote Procedure Call (gRPC), and Basic Station protocols. It forwards uplink messages to Network Servers and schedules downlink messages to the end devices via a LoRa gateway.

Join Server

It is connected to the network server and the application server and has the responsibility of storing the end device’s root keys and handling the OTAA join procedure. It generates and shares session keys with the network server and application server for the secure transmission of LoRaWAN messages.

Network Server

It is responsible for handling the LoRaWAN network layer. It keeps track of the end devices, performs an authentic and integrity check using the MIC algorithm, detects if there are any replayed messages by performing frame counter checks, and sends the message to the appropriate application server.

Application Server

It handles the LoRaWAN application layer which includes decoding and decryption of the uplink message and performs encoding and encryption of downlink messages. It also hosts an MQTT server that exposes the MQTT topic for streaming the application data.

4.4.2 Blockchain and Storage Components

The proposed solution utilizes IPFS and Blockchain as storage.

Blockchain

Blockchain is a decentralized peer-to-peer network that stores firmware metadata and low-powered device information in a smart contract. The main purpose of this component is to enable trust during the firmware update process. Blockchain ensures that there is a single source of truth containing tamper-proof firmware metadata from the manufacturer. This component helps to fulfill some parts of the REQ4 by ensuring the integrity of firmware metadata. It also provides high availability of firmware metadata which fulfills the requirements of REQ5.

IPFS

IPFS is a decentralized peer-to-peer network responsible for storing the firmware image of low-powered devices. It acts as file storage ensuring the high availability of data in the proposed architecture. It particularly ensures the high availability of firmware images which also fulfills the REQ5.

4.4.3 FUS

The important component that this research implemented is the FUS. This component is explained in Table 4.3. The purpose of the component is to provide firmware updates to low-powered devices through Blockchain, providing the low-powered device with a single source of truth, and providing security during the update process.

Table 4.3 FUS Operations.

Tasks	Description
Firmware Request	▪ The FUS handles and manages firmware requests of low-powered devices which can be also initiated by the device owner. ▪ It communicates with the application server via the MQTT protocol and exchanges messages via the topics exposed by the MQTT server.
Connects to Decentralized Networks	▪ FUS is connected to both IPFS and Blockchain networks via HTTPS and the WebSocket. ▪ It runs a daemon that connects with the Blockchain network and handles firmware updates triggered by the manufacturer upon the new upload of firmware metadata.

State Update	It continuously updates the device's progress on the Blockchain during the update process.
Firmware Fragmentation	It performs the firmware fragmentation based on the spreading factor (SF) or the data rate (DR) used by the low-powered device.
Cryptographic Operations	▪ It generates session keys to be utilized for a particular session of the firmware update. ▪ It handles encryption and decryption of sensitive data such as session keys and more. • It performs authentication and integrity check of the sensitive data and firmware image.

The FUS fulfils REQ1, REQ2, REQ3, REQ4, and REQ6. The detailed fulfillment of these requirements is explained in Chapter 5:

4.5 Application Scenario

To understand the proposed architecture, it is important to provide an application scenario to have a better understanding of where it could be applicable. The scenario actors include the manufacturer, Firmware Update Service (FUS), constrained device, Alice as a device owner, and Bob as the attacker.

4.5.1 Scenario Assumptions

- Assume that Alice has planted a garden in a rural area, that is far from where she stays.
- The garden is located in an area where there is no electricity.
- It is assumed that Alice has configured the LoRa devices together with LoRaWAN servers and the FUS, which will be responsible for the entire firmware update process.

4.5.2 Scenario Description

Suppose Alice has a garden and desires to monitor her plants so that she knows their state and treats them according to their need. To achieve this, she buys a battery-powered IoT device equipped with LoRa and Wi-Fi and places it in the garden to monitor the plants. Since the garden is separated by kilometers from her home and planted in an area with no electricity, she cannot rely on Wi-Fi to connect with the device. Alice is worried about the security of the device because people like Bob may intrude on the device to generate misleading data. To overcome this, Alice plans to keep the device up to date with the latest firmware so that the possible vulnerabilities that Bob may use to intrude on the device may be patched. The following sections demonstrate how the proposed architecture can assist Alice to secure her data and have a secure firmware update mechanism.

4.5.3 Illegitimate Firmware Prevention on Blockchain

Suppose the manufacturer has deployed the contract to the Blockchain and it is ready to contract to store firmware data. Let us assume that before the manufacturer releases a new firmware, Bob impersonates the manufacturer by uploading fake firmware. Note that Bob may have a manufacturer's contract address. This contract logic was deployed by the manufacturer on the network and he knows the public key or wallet address of the manufacturer. He then tries to publish the fake firmware to the storage and fake metadata to the Blockchain network and expects that the FUS will feed the malicious firmware to Alice's devices.

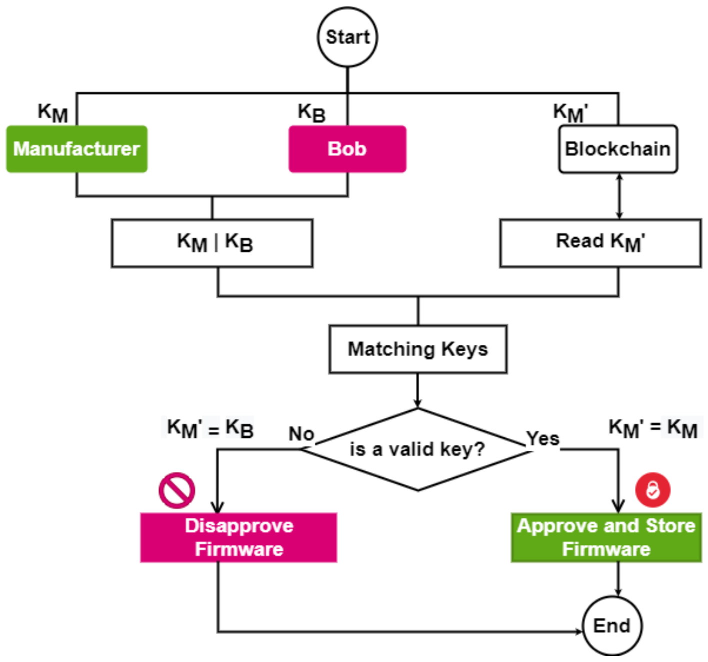


Figure 4.4 Prevention of illegitimate firmware distribution and metadata

Unfortunately, Bob fails to publish the firmware to the network because he does not know the manufacturer's private key and the manufacturer's contract logic allows only the manufacturer to publish new firmware metadata. Figure 4.4 illustrates how this illegitimate activity could be prevented by the proposed solution.

4.5.4 Session Key Eavesdrop

Suppose the manufacturer publishes a new firmware and Alice's FUS generates the session keys and sends them to Alice's devices. During the transmission, Bob eavesdrops on the session keys to use them later. For the next firmware update process when FUS shares session keys with the devices, Bob captures the moment and sends old session keys to Alice's devices that FUS shared on the previous update. However, Alice's devices prevent this kind of attack because the FUS and devices use a function that keeps track of the nonces preventing replaying data. Therefore, it becomes impossible for Alice's devices to accept Bob's session keys to be used during that session of firmware update. Figure 4.5 illustrates how the proposed solution prevents Bob from replaying sensitive data like session keys.

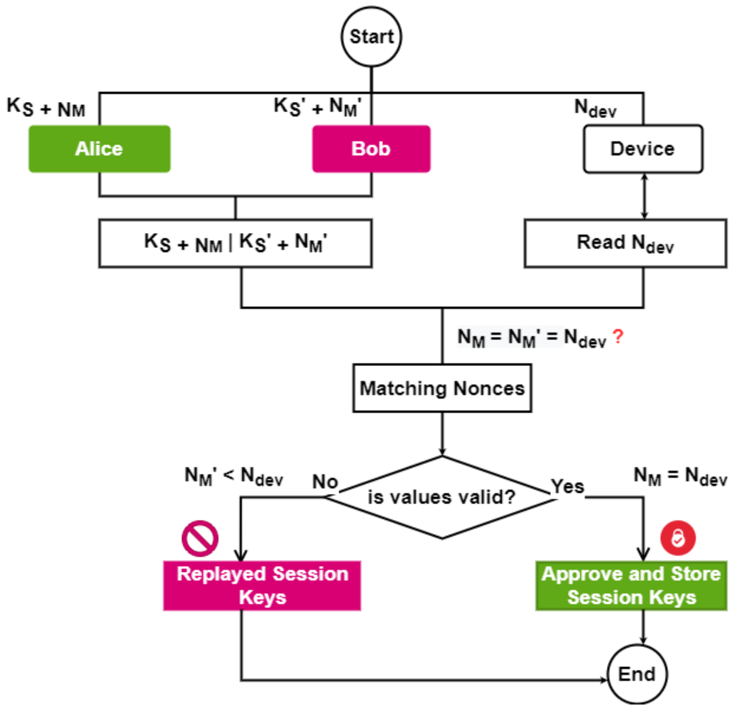


Figure 4.5 Session Key Eavesdrop Illustration

4.5.5 Illegitimate Firmware Prevention on the Device

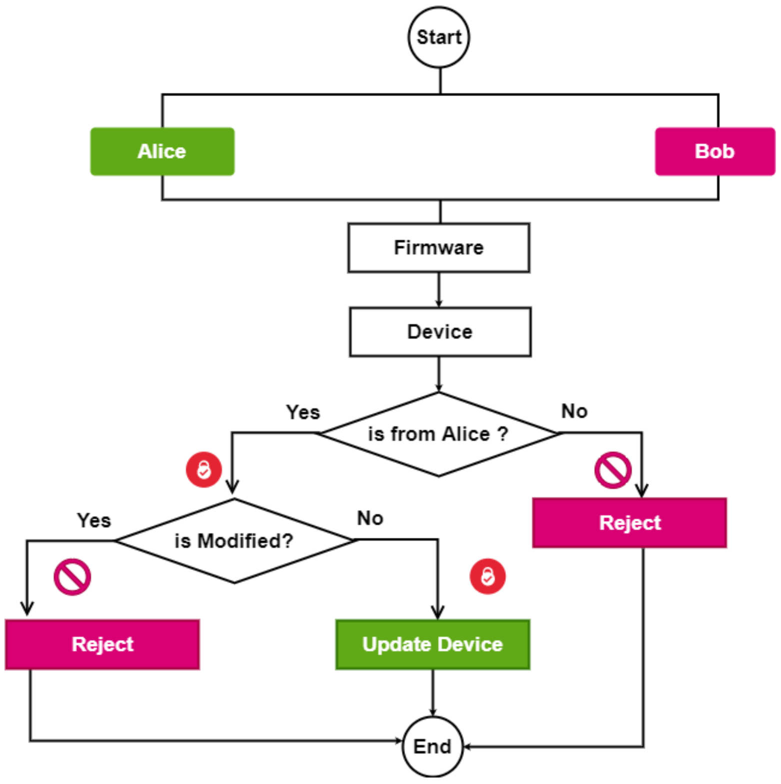


Figure 4.6 Prevention of Bob's Illegitimate Firmware on the Device

After Bob failed to send the session keys to Alice's devices, he somehow found a way to push a fake firmware fragment to Alice's devices. After Alice's devices received all firmware fragments including the malicious Bob's fragment, will then verify their authenticity and integrity. Since one of the fragments came from Bob the verification process will capture that and not install the firmware. This illegitimate activity performed by Bob and its prevention is illustrated in Figure 4.6.

With the proposed solution preventing Bob from performing such activities, Alice can now grow her plants without worrying more about Bob and the security of the devices. In addition, Alice, can monitor the plants from a distance and be able to update the devices in an environment where there is no electricity, with no need for Wi-Fi.

4.6 Security Algorithms

With the requirements and different system components being presented in the previous section, this section focuses on security algorithms utilized by the proposed architecture. These security algorithms mainly focus on achieving data confidentiality, authentication, integrity, and elimination of replay attacks.

4.6.1 Data Authentication

The data authentication proves the origin of the data and ensures that the data has not been modified or fabricated. Data authentication may be achieved using conventional encryption algorithms such as symmetric cryptography or public-key cryptography, also known as asymmetric cryptography. Conventional encryption algorithms can be easily incorporated into low-powered devices unlike asymmetric cryptography and do not require many resources since most of the low-powered ones are limited in resources. Therefore, the proposed architecture utilized symmetric cryptography to provide the data authentication of the firmware image to prove its integrity and authenticity on the end device. Specifically, the MAC is used to determine both integrity and authenticity on low-powered devices.

The FUS component uses an asymmetric cryptography algorithm to provide the authenticity of firmware images at the application layer. ECDSA algorithm provides the authenticity of the firmware utilizing three security keys: wallet address, public key, and private key. The alternative to the ECDSA algorithm would be an RSA algorithm which is commonly used for providing authenticity. The proposed solution however does not utilize RSA, and the reason for this is that some studies like Vahdati *et al.*, (2019) have shown that is more successful in terms of parameters like execution time, energy consumption, memory requirements, decryption time, key sizes, signature generation time, and key generation in constrained IoT devices.

Firmware integrity is achieved through the cryptographic hashing function which is the SHA256 algorithm. A hash function takes an input, breaks it into pieces, mixes them up, and produces a new output. For example, the SHA256 algorithm will take session keys, crumple them, and produces an irreversibly fixed output that can be used to determine the integrity of session keys. Figure 4.7 shows how SHA-256 works.

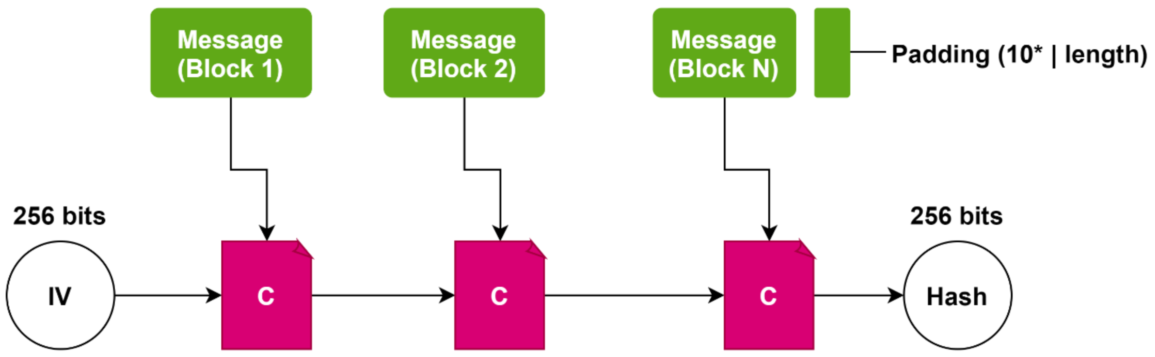


Figure 4.7 SHA256 hashing algorithm

With the SHA256 algorithm, the input value is divided into elements of equal length. These are called blocks. Since a multiple of the block length is required, it is usually necessary to fill in the data, that is, expand it. The padded value is the padding. Then, the processing is done in blocks. The blocks are executed and used as a key for intermediate calculations on the data to be encoded later. The result of the last calculation is the output value is the hash value. SHA256 is the most-used hashing algorithm. There are alternative algorithms such as MD5, and SHA1 however, collisions have been found with these algorithms (Stevens *et al.*, 2017) but no collisions have been found yet with SHA256 algorithms.

SHA256 is used in the MAC algorithm to help in providing not only the integrity but also the authenticity of data. As mentioned before, the proposed solution utilizes the MAC algorithm specifically, the HMAC-SHA256. Although there are other types of MAC algorithms apart from the HMAC-SHA256, it is the proposed architecture because it provides integrity and authenticity through hashing. The hashing functions are faster than block ciphers which other MAC algorithm like Cipher-Block Chaining Message Authentication Code (CBC-MAC) uses to provide integrity and authenticity (Kaliski, 2011).

4.6.2 Secure Distribution of Data Security and Replay Attack Prevention

The authentication and encryption of data are not enough in distributing the firmware securely. The communication between a sender and the receiver (for instance, between the FUS and the low-powered device) can be taped in by the attacker. The attacker could break the communication to uncover the plaintext from the ciphertext or try to discover the encryption key to attempt to decrypt the data sent between the sender and receiver in the future. The attacker can try to record the data being exchanged and then replay the old data to the low-powered device. Note, that the sent old data will be interpreted correctly by the low-powered device since it was encrypted by the FUS using the shared secret key. Figure 4.8 illustrate how a replay attack can happen between the FUS and the low-powered device.

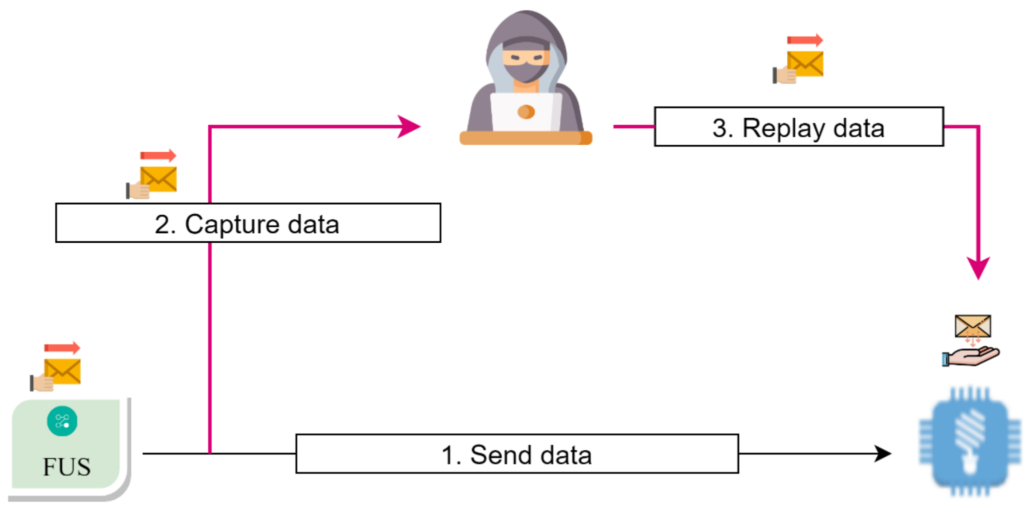
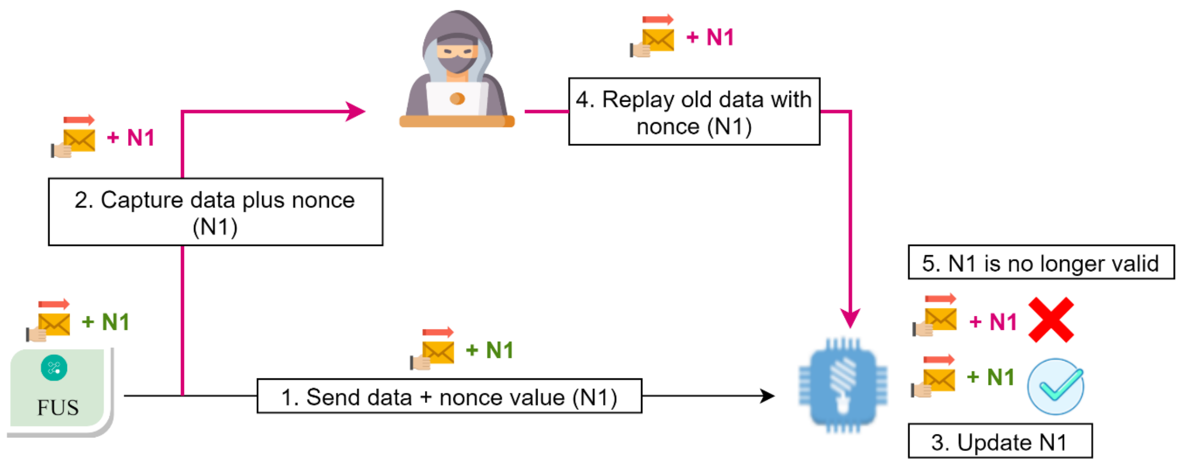


Figure 4.8 Replay attack in FUS component

To avoid the replay attack in the proposed system, the architecture utilizes the nonce values. The nonce values must be:

- A number that is used only once
- Different for each request
- Difficult for an attacker to guess

An example of the nonce values could be a timestamp, counter, random value, etc. Since most of the low-powered devices do not have an internal clock and are not directly connected to the Internet, the timestamp cannot be utilized.



+ N1
4. Replay old data with nonce (N1)
+ N1
2. Capture data plus nonce (N1)
5. N1 is no longer valid
+ N1
FUS
+ N1
1. Send data + nonce value (N1)
+ N1
+ N1
3. Update N1

Figure 4.9 Replay Attack Illustration

The architecture uses the counter as a nonce value to prevent a replay attack. This is illustrated in Figure 4.9. The counter values are generated by the function f(n), where f(n) is a function that increments the nonce value. Both FUS and the low-powered device must know this function. When the FUS receives N1, it uses the function f(n) to increase the nonce and generate N2 which are both sent along with the encrypted data. The device will then decrypt the data with the nonce values using the shared secret key. The correctness of the nonce values will be verified by the low-powered device which will then send an increment to the N2 using f(n) and send it to the FUS.

4.6.3 Data Confidentiality

LoRa stack provides confidentiality of data through AES. This means every low-powered device runs an AES algorithm to encrypt and decrypt incoming and outgoing LoRa payloads. The proposed architecture is designed to achieve confidentiality through the AES algorithm. The counter mode of operation (CTR mode) is used for the encryption and decryption of data.

4.7 Proposed Blockchain Smart-Contract Operations

At this point, the security algorithms responsible for securing the update process have been presented. In the following sections, the focus is mainly on describing the interaction between the system components. But before the overall interaction/procedure is described, there is a need to introduce an important component of Blockchain which is a smart contract. Two contracts were designed, and each has its operations. The first contract is a manufacturer’s contract which stores firmware metadata to be used during the update process. The second contract is a FUS contract that stores the low-powered device's data such as the device's unique ID, the manufacturer’s contract address, device model, current firmware, and the status. The manufacturer’s contract is publicly stored and can be utilized by any entity to retrieve firmware metadata and check for the availability of the new firmware. Figure 4.10 illustrates the operations from both contracts.

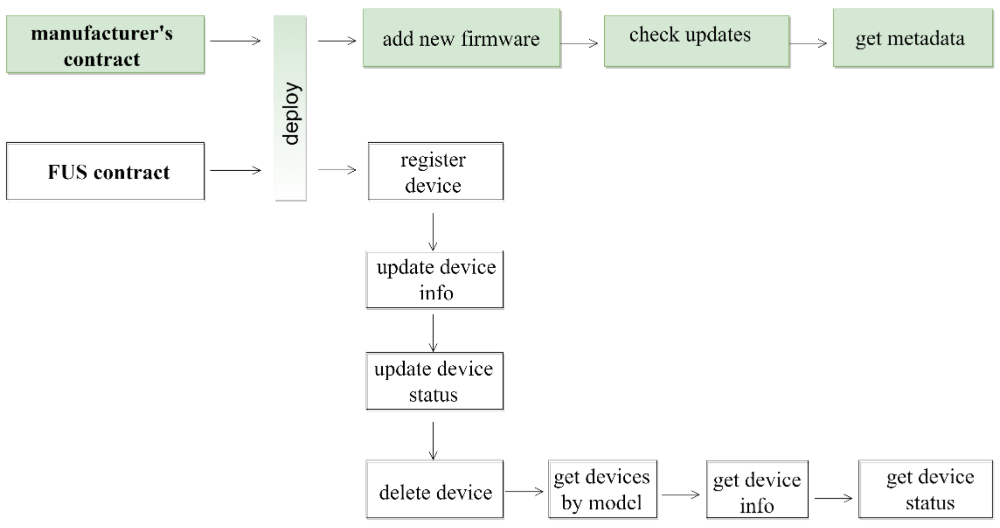


Figure 4.10 Smart Contracts Operations.

4.7.1 Manufacturer Smart Contract

The manufacturer's smart contract has four Blockchain operations that include deployment of the contract, adding new metadata, checking updates, and retrieving metadata. These operations are explained in this section together with their pseudocode.

Deployment

The manufacturer must first deploy the contract to the Blockchain network before any firmware update process occurs. When the contract is deployed, there is an algorithm responsible for assigning the manufacturer's contract address as an address that owns a contract in the network. This gives the opportunity in the future to know who created and deployed the contract and to restrict access to sensitive data. The pseudocode of this algorithm is represented in Algorithm 1.

Algorithm 1: Pseudocode for assigning manufacturer's contract address and name to Blockchain

```
Input: deployer's contract address and name
Result: stores the deployer's contract address to storage variable
string manufacturer_id;
string manufacturer_name;
function constructor (string id, string name) do
    Add id, name to manufacturer_id and manufacturer_name storage variable;
end
```

Adding New Metadata

Once the contract is deployed on the network. The manufacturer can add the new metadata. Adding new metadata is a crucial operation that must not be allowed to be executed by any other entity in the Blockchain but only the manufacturer. Therefore, Algorithm 2 has to check first who calls it before it accepts the metadata. The newly uploaded metadata is saved on the metadata list structure and is represented in the form of key-value pair.

Algorithm 2: Pseudocode for adding new metadata

```
Input: model, version, firmware metadata represented in key-value
Result: Firmware record is updated with a new metadata
mapping (string => Firmware Metadata) metadata_list;
if msg.sender == firmware Provider then
    loop through the entire model_list upload and check if the version of the model exists
    emit new Firmware (Firmware Details);
else
    error: Not authorized for such operation;
end
```

Checking Updates

After the metadata has been uploaded, anyone who wants to check for newly uploaded metadata can interact with Algorithm 3. For example, The FUS will call this function on the manufacturer’s contract to check if there are any new firmware updates available for the provided current version of the device and the provided device’s model. The algorithm returns true or false based on the availability of the firmware, meaning true is returned if the device’s current version is less than the version that exists on the Blockchain.

Algorithm 3: Pseudocode for firmware availability check

```
Input: model, current_version
Result: Returns true or false based on the firmware availability
mapping (string => Firmware Metadata) metadata_list;
string [] public model_list;
uint256 i;
for an available model in model_list do
    if model == model_list[i] then
        if metadata_list[model.version] is greater current_version then
            return true;
        end
    end
    increment i
    return false;
end
```

Download Metadata

Algorithm 4: Pseudocode for retrieving the firmware metadata

```
Input: model,
Result: Return metadata
mapping (string => Firmware Metadata) metadata_list;
uint256 i;
for an available model in model_list do
    if model == model_list[i] then
        return metadata_list[model].metadata
    end
    increment i
end
```

If the availability state of the firmware update returned by Algorithm 3 results is true, Algorithm 4 will return the latest firmware metadata when is called. The FUS calls this operation to get the new metadata for the low-powered devices. The algorithm first checks if the provided model is valid before it returns the metadata. If this is not checked, this will result in using the invalid key on the metadata list and lead to confusion.

4.7.2 The FUS Smart Contract

In Section 4.7.1, the manufacturer's smart contract operations were explained, this section presents the FUS contract operations. which is owned and managed by the owner of the low-powered IoT device. This contract stores, updates and retrieves low-powered device information. It has several operations that can be invoked, including device registration, updating the device, and getting the device information. This section explains and presents these operations.

Deployment

The device owner (the FUS owner) needs to deploy the contract to the Blockchain network so that it can store information on the low-powered devices. When the contract is deployed, this function stores the owner’s contract address and the name of the owner on the Blockchain network. The owner’s address can be used later to restrict access to some of the contract operations. For example, the FUS owner is the only entity allowed to update device information. Algorithm 5 is similar to Algorithm 1 whereby the manufacturer's contract deploys the contract on the Blockchain network.

Algorithm 5: Pseudocode for assigning device owner's contract address and name to Blockchain

```
Input: owner's contract address and name
Result: stores the owner's contract address to storage variable
string owner_id;
string owner_name;
function constructor (string id, string name) do
    add id, name to owner_id and owner_name storage variable;
end
```

Device Registration

Algorithm 6: Register a new LoRa device

```
Input: device details
Result: Updated device list record
mapping (string => string[]) devs;
mapping (string => Devices[]) devicesList;
string[] public deviceIDList
 if msg.sender == updateServiceOwner then
        for available device in deviceIDList do
            if devID == deviceIDList[i] then
                exist == true;
            end
        end
        if not exist then
                add a new device to the deviceList
        end
 else
        Error: Not authorized for such operation;
end
```

After a successful deployment, the device owner needs to register low-powered devices on the network since the FUS component needs to know which Low-powered devices need to be updated. The registration operation is crucial because it must not be called by anyone other than the FUS owner. The device information includes the LoRaWAN device application id, the device id, the manufacturer’s smart address, wallet address, model, current version, and the device update status. The registration is handled by Algorithm 6.

Update Device Information and Status

The device information on the Blockchain can be updated together with the update status. The updated information includes the LoRaWAN device application id, the device id, the device manufacturer’s smart address, the wallet address, the device model, and the device version. The status update keeps track of the update process of the device, e.g., the status of the session keys whether were exchanged or not, and the number of firmware fragments sent. Keeping the state helps in case of any interruptions during the update process. Algorithm 7 and Algorithm 8 illustrate the pseudocode for updating device information and device status respectively.

Algorithm 7: Update Device Information

```
Input: devID, device information
Result: return true for the device's successful update and false if device information is not updated
mapping (string => Devices[]) devicesList;
 if msg.sender == updateServiceOwner then
        if devicesList[devID].exist do
              update device information
              return True
        end
 else
        Error: Not authorized for such operation;
end
```

Algorithm 8: Update Device Status

```
Input: devID, new device status
Result: updated device status
mapping (string => Devices[]) devicesList;
 if msg.sender == updateServiceOwner then
          update devicesList[devID].status with a new device status
 else
        Error: Not authorized for such operation;
end
```

Delete device

Algorithm 9: Delete low-powered device

```
Input: devID
Result: delete device from the list
mapping (string => Devices[]) devicesList;
 if msg.sender == updateServiceOwner then
     if devicesList[devID].exist do
         delete a device with matching devID
         return True
     end
 else
     Error: Not authorized for such operation;
end
```

Apart from updating the device on the Blockchain, the device owner can also delete the device if it is no longer needed. This operation will only be executed by the owner of the FUS contract and if another entity tries to execute the function, it will fail because it requires the address of the owner for successful execution. The delete operation is illustrated by Algorithm 9 pseudocode.

Get Device(s) Information Operations

The owner's contract has three algorithms that get information about the device on the network. These algorithms include the retrieval of devices by the model's name, getting the device information, and the device update status and are illustrated by Algorithm 10, Algorithm 11, and Algorithm 12 respectively.

Algorithm 10: Get devices by Model

```
Input: model
Result: Returns devices with matching model
mapping (string => Devices []) devicesList;
function getDevicesByModel(string model) do
    return devicesList[model]. model;
End
```

Algorithm 11: Get device information

```
Input: devID
Result: Returns device update status
mapping (string => devicesInfo []) devicesInfo;
function getDevStatus (string devID) do
```

```
    return devicesInfo[devID];
End
```

Algorithm 12: Get device update status

```
Input: device ID
Result: Returns device update status
mapping (string => Devices []) devicesList;
function getDevStatus (string devID) do
    return devicesList[deviceID].status;
End
```

4.8 Overall Procedure of the Proposed Architecture

This section explains the interaction between the system components during the firmware update process. The interaction is classified into four main phases: firmware upload, device registration, firmware initiation, firmware download, and firmware verification. In each phase, the interaction is explained with the security measures taken.

4.8.1 Firmware Upload Phase

The firmware upload phase occurs after the manufacturer's contract has been successfully deployed to the Blockchain network and after the manufacturer has successfully connected to the IPFS network.

The firmware upload involves the interaction between three main system components the device manufacturer, the IPFS network, and the Blockchain network. The objective of this phase is to have a firmware image successfully deployed on the decentralized IPFS network and the metadata stored on the Blockchain network. The manufacturer connects to the IPFS node to publish the new firmware image to the IPFS network and also connects to the Blockchain node that is synced with the network to publish the metadata. After the firmware and metadata are deployed via the connected nodes, they are synced with the rest of the network. Figure 4.11 illustrates this process of uploading the firmware and metadata to the Blockchain and IPFS network.

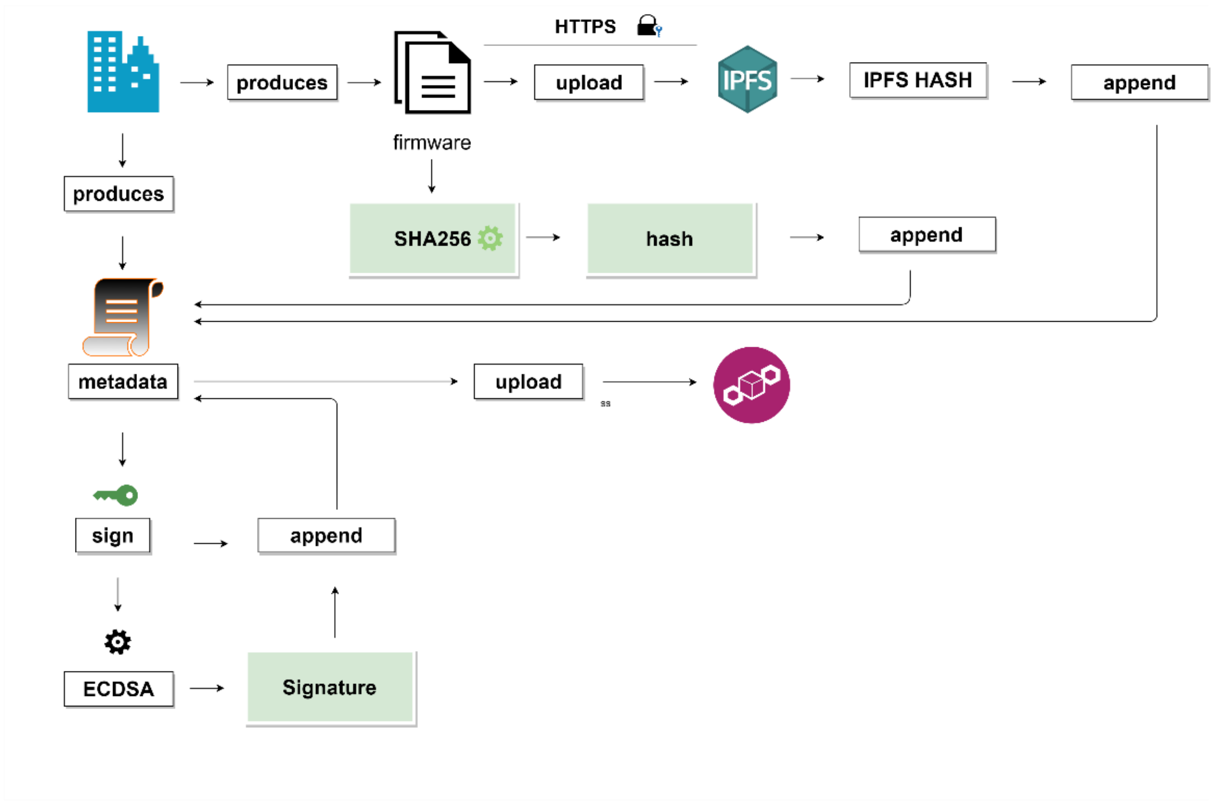


Figure 4.11 Firmware Upload Procedure.

The metadata that is deployed on the Blockchain is structured as follows:

```
{
    name: 'LoPy4-firmware',
    version: '1.2.0',
    model: 'LoPy4',
    SHA256: '569948b4baa...',
    IPFS_HASH: 'QmaY7aKo...',
    Signature: 'Ed30Ac8a...',
    ....
    ....
    ....
}
```

Figure 4.12 Structure Example of the Metadata

Figure 4.12 shows the metadata that is deployed by the manufacturer to the Blockchain. This metadata is constructed or created from the web application this is illustrated in the implementation chapter.

Security Measures: During Firmware Upload Phase

Security measures need to be taken when the firmware is uploaded to the networks. Thus, this section explains the security involved in the firmware upload phase. During this phase, it is very important to verify when the uploaded firmware is legitimate before it gets stored. The illustration of how the verification process of firmware metadata is done is shown in Figure 4.13:

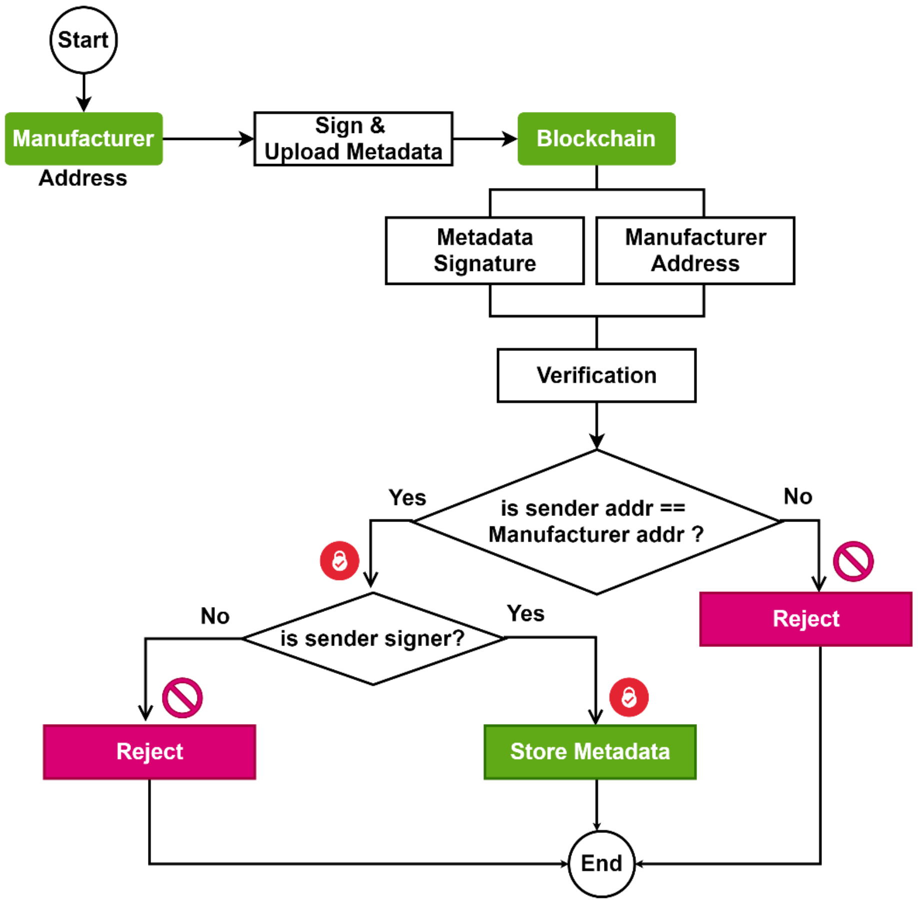


Figure 4.13 Verification process of firmware metadata

- Integrity - The firmware image can be modified during transmission. Therefore, the manufacturer hashes the firmware image using the SHA-256 algorithm to prevent any alteration that could take place in the update process. The calculated SHA-256 hash forms part of the metadata.

- Authentication - The manufacturer needs to sign the firmware to prove the ownership digitally. The manufacturer uses the private key to sign the metadata. The Elliptic curve digital signature (ECDSA) signature is produced and appended to the firmware metadata. By appending the signature to the metadata, it will be easy to verify the authenticity of the firmware image. Moreover, this enables metadata to be immutable and tamper-proof since it is on the Blockchain network.

- Firmware Availability - The firmware image is deployed on the manufacturer's IPFS node that syncs with the IPFS network. The other IPFS nodes on the network will sync with the uploaded firmware; this ensures the high availability of the firmware image even if the manufactures node is unavailable on the network.
- Authorization - The firmware metadata describes the firmware images. It consists of the integrity hash, the manufacturer's digital signature, the firmware's size, the firmware version, the location of the firmware, etc. Firmware metadata plays a considerable role during the verification process; therefore, no other entity apart from the manufacturer is allowed to deploy the firmware metadata. The Blockchain contract enforces authorization only, allowing the manufacturer to be the only entity of the network to upload firmware metadata.

4.8.2 Registration Phase

Registration is required for the low-powered device to be a part of the LoRaWAN network. This includes the generation of keys that the device will use during the join procedure. It is assumed that the end device is already configured to join the LoRaWAN network in this phase. This phase particularly describes the device registration to the Blockchain network. It involves interaction between three system components: the device owner, the FUS, and the Blockchain.

The owner registers the LoRa device to the Blockchain network via the FUS. The FUS directly connects the owner with the Blockchain network to manage the devices. Device management includes registering the device, deleting the device, and updating the device information. Note that for the FUS to register the device invokes the contract operation illustrated in Algorithm 6. This phase ensures that the device is successfully registered and is ready to receive a firmware update.

Security Measures: During the Registration Phase

The FUS exchanges device information with the Blockchain during the registration phase. However, the exchanged information is stored securely on the public Blockchain, even though the device's information is immutable and tamper-proof on the network. It is required that the FUS encrypts the information before it is stored in the Blockchain. Therefore, the confidentiality of the data must be ensured. The FUS ensures the confidentiality of LoRa device information through the Advanced Encryption Standard (AES). The encryption of this information is done utilizing the Counter Mode (CTR) as a mode of operation. FUS uses the shared secret key (K_{FUS})of 128-bit for both encryption and decryption of the Blockchain data and is responsible for generating the master key (K_M) for each registered device. The K_M is used in AES to provide confidentiality of messages between the FUS and the end device. This key should be kept secret between these entities. Figure 4.14 demonstrates the security activities for this phase.

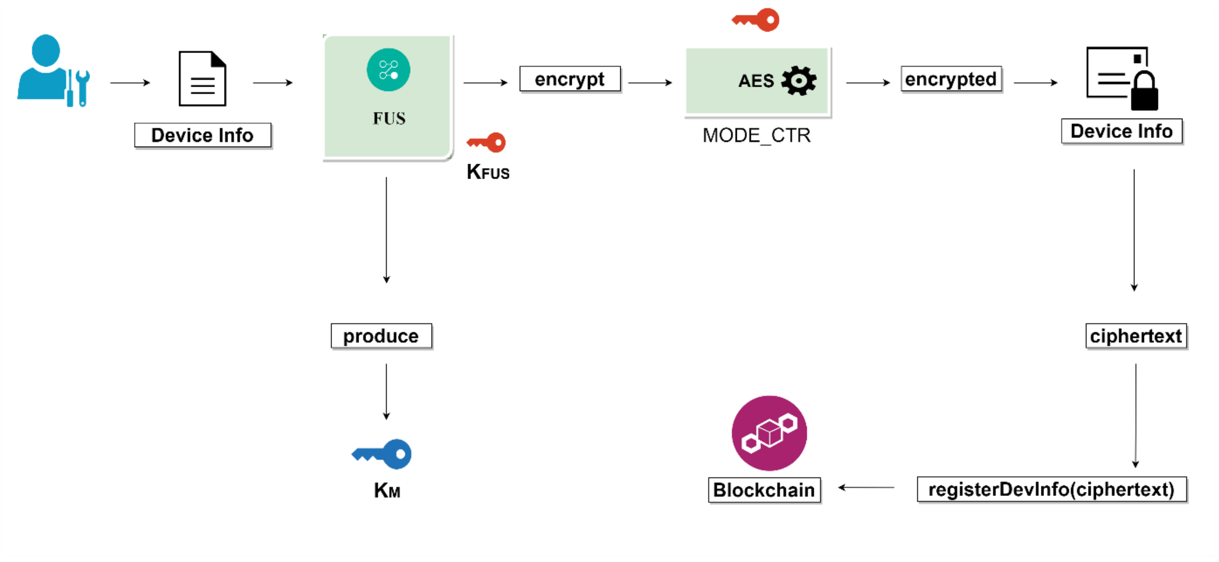


Figure 4.14 Device registration phase

4.8.3 Initialization Phase

At this point, the devices are registered and ready to be updated. This phase talks about how the firmware process starts and what entities are responsible for it. The firmware update process could be started by two entities or triggered in two different ways. Firstly, it could be triggered by the device owner since he is responsible for managing the end device. The second way is based on the Blockchain event which is triggered by the device manufacturer when uploading the new firmware metadata on the Blockchain network. This phase aims to initialize the firmware update process and then successfully exchange the session keys between the FUS and the low-powered device(s).

The device owner triggers the firmware process by communicating with the FUS via a command-line (CLI) script which implements the MQTT protocol. The FUS exposes the MQTT topic, which listens to the device owner's firmware request. The FUS will be then responsible for the entire update process afterward. The session keys will be generated by the

FUS and exchanged with the end device via the LoRaWAN network. The session keys will not only be exchanged but the security also needs to be considered since they are a crucial part of the firmware update process. After successfully delivering the session keys, the end device performs a security check on them before they are utilized. The end device also sends the uplink message for confirming the successful delivery of session keys.

Security Measures: During Firmware Initiation Process

Before the session key exchange occurs, both the end device and the FUS must have shared the secret key in front. The shared secret key K_M is the one that was generated earlier by the FUS during the device registration phase. The FUS prepares the session key message shown in Table 4.4. The session key message is formatted as follows:

Table 4.4 Session Key Message Exchange

	ID	IV Nonce	DevNonce	ServNonce	Mode	DR	AESSKey	MACSKey	TAG
Bytes	1	4	3	3	1	1	16	16	4

- ID - The id uniquely identifies the message.
- IV nonce - IV nonce is used during the encryption and decryption process of session keys.
- DevNonce and ServNonce - DevNonce and ServNonce are used to prevent replay attacks between the FUS and the device**.**
- Mode - This is a LoRaWAN device class mode. E.g., Class A, Class C, and multicast.
- DR - The data rate to be used to update the device e.g., DR0, DR1, etc.
- AESSKey and MACSKey - Security keys generated for the particular session of the firmware update process.
- TAG – Refers to the signature or tag to be used by the end device to verify integrity and authenticity.

The FUS generates the session keys (Ks) AESSKey and MACSKey using K_M which are the two keys used to determine the confidentiality, integrity, and authentication of the messages during the particular firmware update session. One of the session keys being exchanged is the AES session key (AESSKey), which provides confidentiality of sensitive messages such as MAC tags and nonce values. The FUS randomly generates AESSKey and MACSKey, updates the nonce value of the end device using an incremental function, and finally generates its nonce value N_2. The session key exchange message is encrypted using the K_M.

When the end device receives the session key data, it decrypts it using the same shared secret key K_M and then checks if the data has not been replayed by checking the nonce value N_1 received. As a response to the received session keys, the end device sends an acknowledgment message with the updated values of N_1 and N_2 Encrypted with the recently shared session keys. The FUS receives the message and decrypts, checks for any replay attack, and updates N_1 and N_2. The firmware update process can also be initiated by the manufacturer's Blockchain event on adding new firmware metadata on the Blockchain. The firmware event initialization may work very well when updating a set of the end device because it enables FUS to look for all devices that match this new metadata. The process of session key exchange and detection of replay attacks is still the same as shown in Figure 4.15.

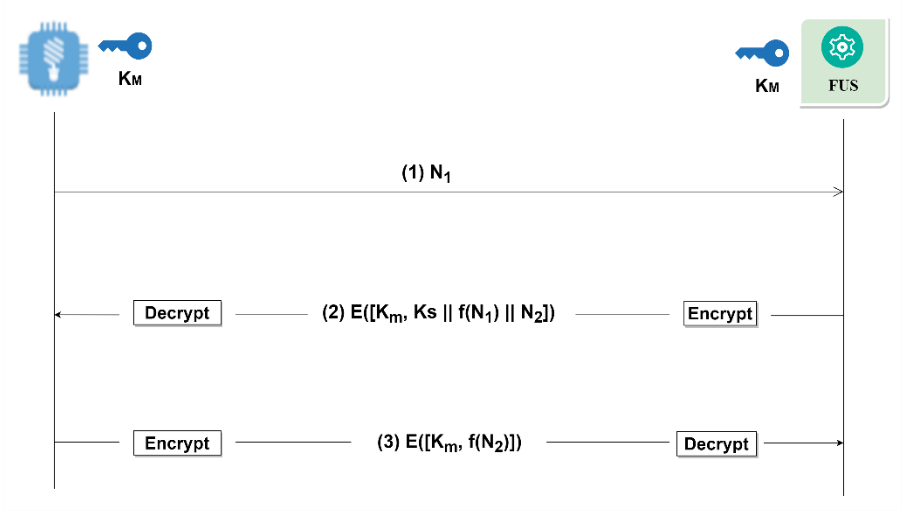


Figure 4.15 Session Key Exchange.

During the session key exchange, the architecture provides confidentiality, and data authentication and protect against replay attack. These security properties are clearly explained as follows:

- Replay Attack - The nonce values are randomly generated to prevent replay attacks and must be only used once during the firmware update session. The FUS and the device have a function that keeps track of these values and checks for any possible replay attack on each message sent, i.e., session keys.
- Confidentiality - The architecture utilizes AES to provide confidentiality of session keys. It is recommended that the encryption keys be changed over time. Therefore,

The FUS generates these session keys instead of using the K$_M$ for both encryption and decryption.

- Data Authentication - The second key is the MAC session key used for providing the integrity and authentication of the message

4.8.4 Firmware Download

This phase of the firmware update demonstrates what happens after the session keys were successfully exchanged. After successfully exchanging the session keys, the FUS requests firmware metadata on the Blockchain network and the firmware image on the IPFS network.

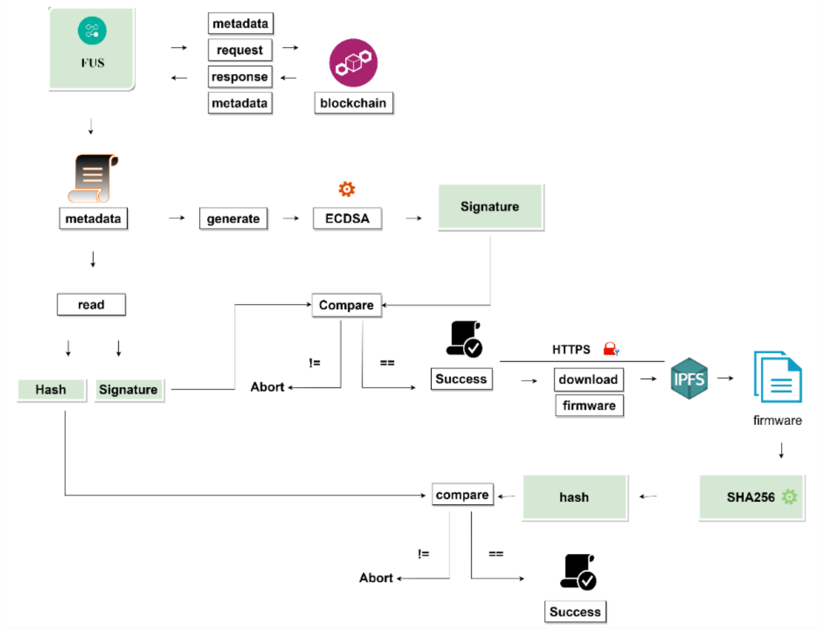


Figure 4.16 Firmware Downloads and Verification Phase.

The FUS is connected via a secure channel (HTTPS) both on the IPFS node and Blockchain node. The firmware authenticity must be achieved at this phase to ensure that the right firmware will be sent and updated by the end devices.

Security Measures: During Firmware Download Phase

After the firmware has been successfully downloaded from the IPFS file storage, it needs to be verified against any malicious activities. This includes alteration and determining its authenticity. Figure 4.16 shows the download process, and Figure 4.17 further illustrates how both authenticity and integrity are achieved.

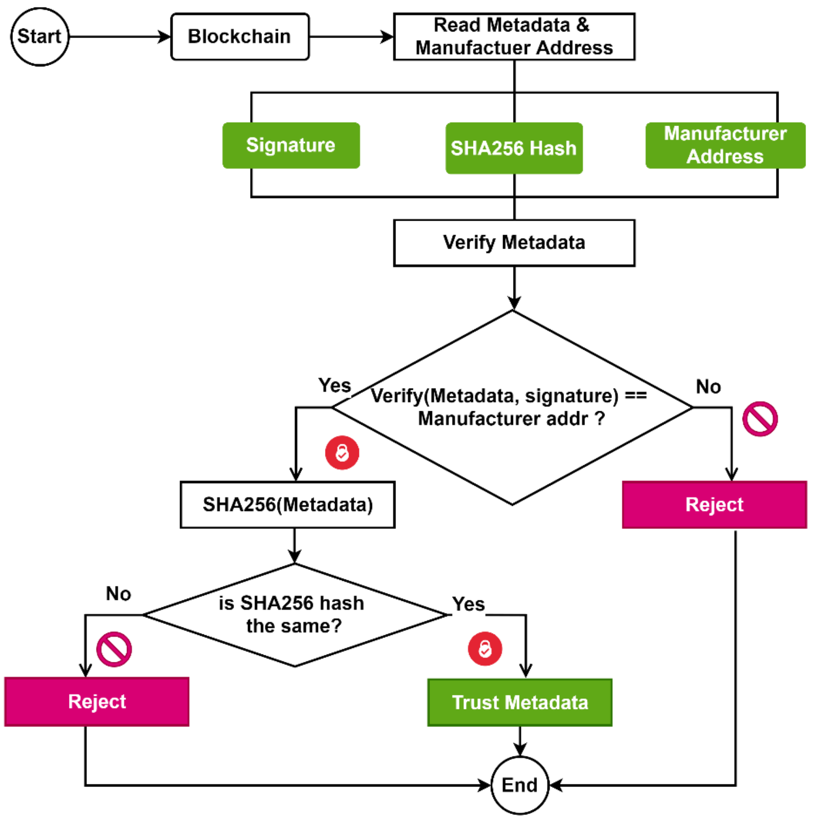


Figure 4.17 Firmware Downloads

- Authentication - In the previous phase of firmware distribution, the manufacturer had signed the firmware metadata and uploaded the metadata to the Blockchain network. Now in this phase, the manufacturer's digital signature is utilized to prove the authenticity of the metadata. The manufacturer has three important keys on the Blockchain network: the private key, the public key, and the wallet address (K_{MW}). The private key is used to sign the firmware and must be kept secret. The wallet address is a hashed public key and is allowed to be shared with other entities on the Blockchain network. The wallet address plays a huge role in determining the authenticity of the firmware in the update process. The proposed architecture uses a function that takes the ECDSA digital signature with the metadata to produce the wallet address that signed the firmware metadata. The produced wallet address is matched against the wallet address registered earlier in the registration phase by the device owner. If both addresses match, the metadata does come from an authentic source and can be used to download the firmware image.

- Integrity - Regardless of the secure channel between the IPFS and the FUS, firmware integrity has to be achieved. The FUS obtains the firmware image and recomputes the SHA-256 hash which is then compared with the SHA-256 hash of the metadata. If both hashes are the same this confirms that the firmware image has not been altered in transmission.

4.8.5 Firmware Data Authentication

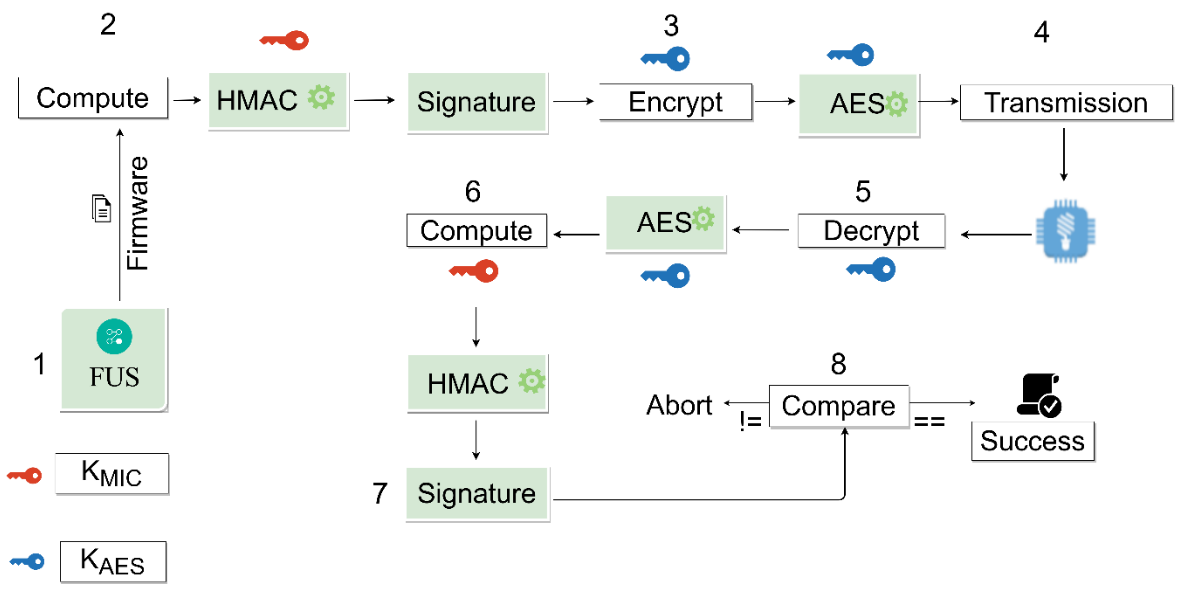


Figure 4.18 Firmware Verification on the End Device.

After a successful firmware verification, the firmware image is ready to be sent over to LoRaWAN. This section describes how the firmware image is secured and explains the verification process shown in Figure 4.18 that the end device performs.

The FUS performs fragmentation based on the spreading factor (SF) and the end device's region. The MAC of the firmware is first calculated using the HMAC-256/CMAC algorithm and sent over to the LoRaWAN so that the end device can verify both the integrity and the authenticity of the firmware. Usually, the digital signatures based on the public and private keys are used to verify the authenticity and the integrity of the firmware image on the end device, however, since these devices are limited in storage, some cannot incorporate digital signatures because they require more processing power to do the verification. Most of the constrained device's symmetric cryptography is considered lightweight, even LoRaWAN is based on symmetric-key cryptography to determine the authenticity and integrity of the data. Therefore, the proposed architecture adheres to the current cryptographic technique provided by LoRaWAN to deliver firmware updates to the end device via cryptographic technique. Note that asymmetric cryptography is used at the application layer to ensure the firmware's authenticity and integrity before sending it over to the LoRa.

Security Measures

Figure 4.18 shows how the verification process is done by the low-powered device when the firmware is received but to further clarify this process, Figure 4.19 is presented to illustrate how the confidentiality and authenticity of the firmware update are achieved between the FUS and the low-powered device.

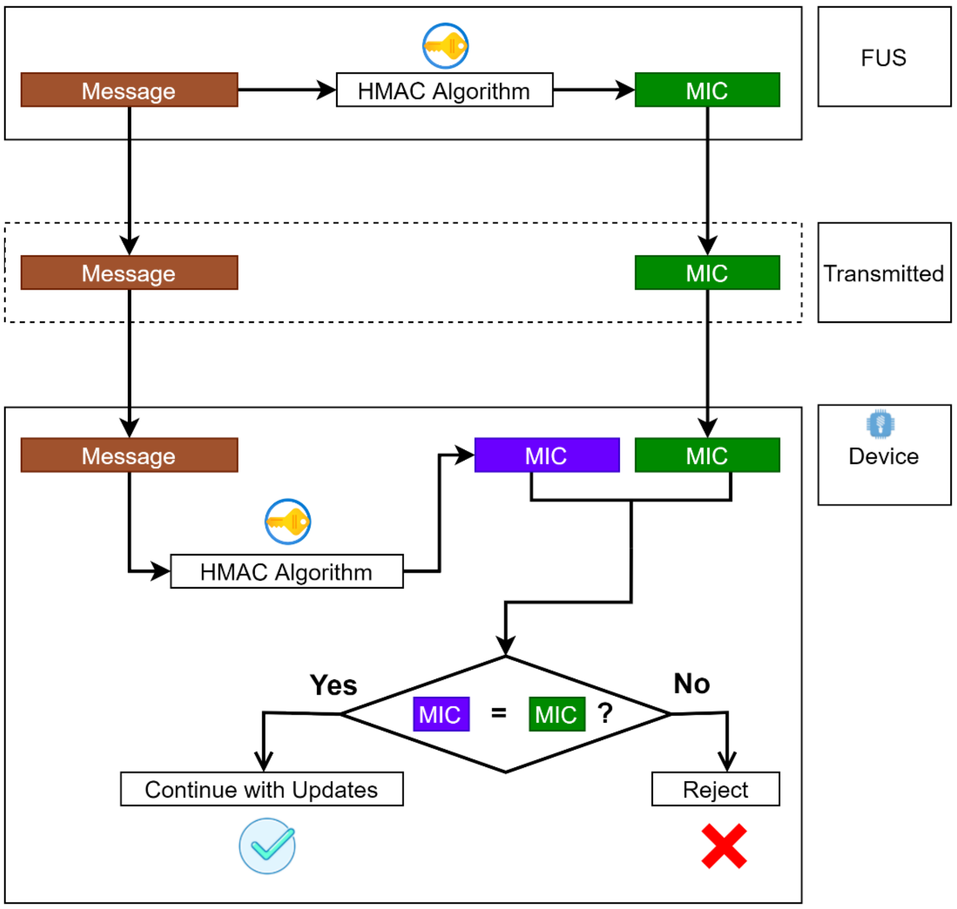


Figure 4.19 Confidentiality and Authenticity of FUS and the Device

- **Confidentiality** - The MIC needs to be sent encrypted over the channel; hence the MIC is encrypted with the session-shared secret key K_S. The end device receives the MIC and decrypts it with a similar AES session key. The end device uses the CTR mode when decrypting the MIC as the data was encrypted using the same mode of operation.
- **Authentication and Integrity** - After the device has received all the firmware fragments including the missing ones, it needs to determine whether the firmware comes from the authentic source and has not been changed on the transmission. Figure 4.18 and Figure 4.19 demonstrate this process of verification where authentication is achieved via the MIC algorithm.

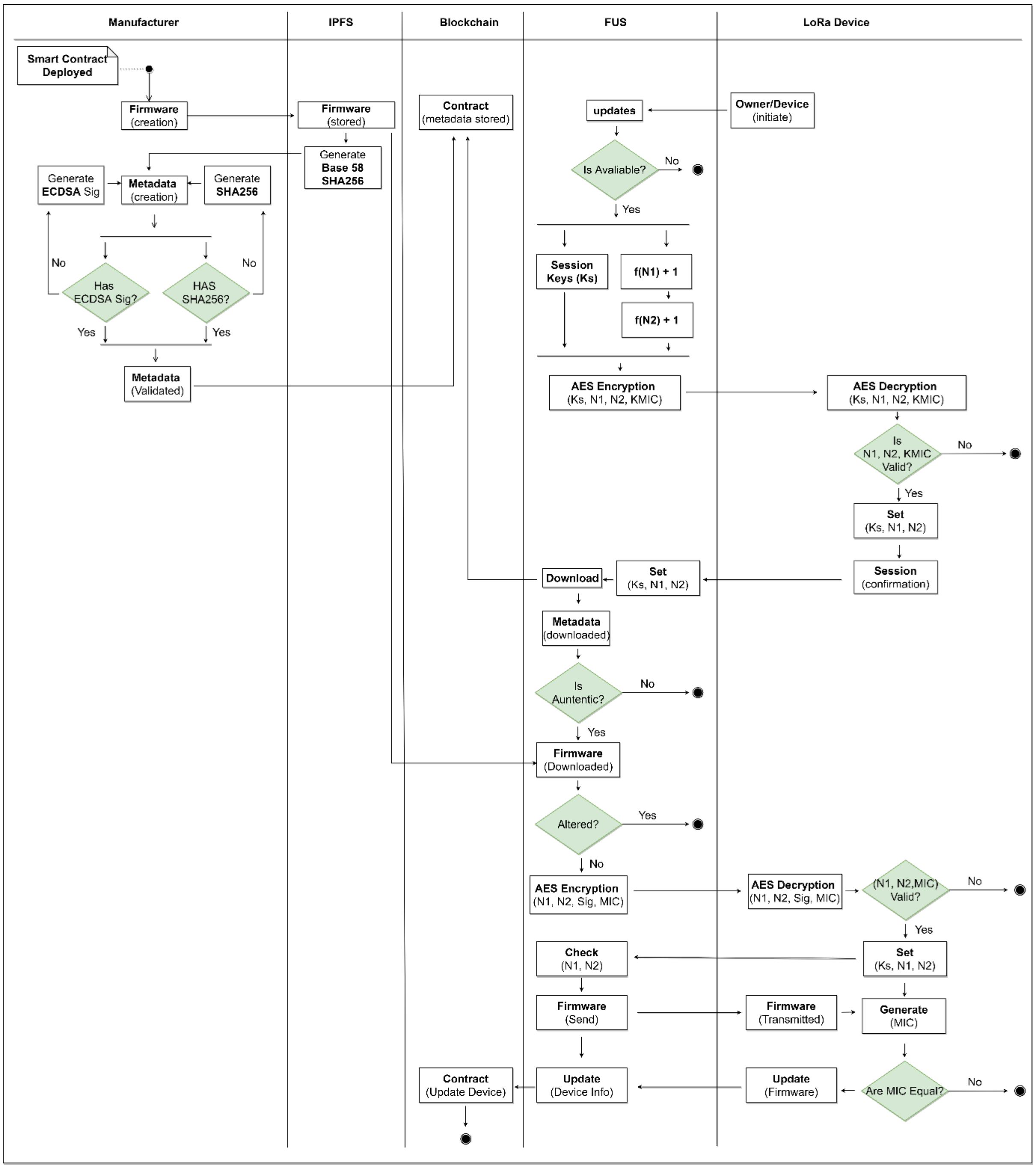


Figure 4.20 Security Activity diagram.

To summarize all the phases and to give a clear overall interaction, the activity diagram is utilized. The activity diagram visualizes the data flow behind the proposed architecture as illustrated in Figure 4.20. It also clearly describes how each component interacts with the other while representing the data flow.

The data flow begins with a manufacturer providing the firmware and metadata as input to the system. Firmware and metadata are deployed on the IPFS network and Blockchain network respectively. Firmware metadata is validated to ensure it consists of the necessary information required to determine its origin and integrity. The ECDSA and integrity hash are checked and once successfully checked, the validated metadata data is produced and it is sent to the Blockchain. The FUS then checks updates or gets notified via event as mentioned in the initialization section.

After the successful update, necessary session keys, signature validation, integrity check, and replay attack checks are done between the FUS and the Low-powered device. If all firmware checks are successfully performed, the device can flash the firmware into memory, and once done, the device status about the newly installed firmware is updated in the Blockchain.

Chapter 5: Implementation

This chapter details the implementation process of the proposed architecture that was discussed in Chapter 4:. A quick overview of the development tools and programming languages utilized is provided. This is followed by the implementation of the Long-Range Wide Area Network (LoRaWAN) network and Long-Range (LoRa) nodes used in this study.

5.1 Blockchain Framework

The Blockchain can either be public or private. The public Ethereum Blockchain framework was chosen for the implementation of the Blockchain-based firmware update architecture proposed in this study. Table 5.1 provides a comparison of possible Blockchain that can be utilized to securely deliver firmware updates to the devices. Our choice of Blockchain is also justified.

Table 5.1 Blockchain Comparison

	Public Blockchain	**Private Blockchain**
Anonymity	Public Blockchain provides anonymity by establishing the user identity utilizing addresses. Public Blockchain is handy when developing solutions that do not require any knowledge of the user identity. Thus, user identity is of no importance when it comes to the firmware update.	The private network requires true user identity since it may want to grant access to only specific users. It can be noted that in the firmware updates process, all users have equal access, thus, a private network may not be practical in this case.
Transparent	The data is fully transparent enabling anyone to have access to it. Note firmware updates are open and allow open downloads for device owners.	The private network has low transparency since some of the data may be not fully visible to the user.
Decentralized	The public network is controlled by multiple entities that validate and verify the transactions in the network.	Only a certain group of people or an organization owns the network. Note this can lead to the single-point-of-failure which may not be a

		desirable attribute for firmware updates.

The reasons for the choice of the public Blockchain, particularly the Ethereum Blockchain are:

- The firmware is usually shared publicly with the device owners via websites, blogs, etc. Therefore, Ethereum as a public network was chosen because it allows anyone to be a part of the network.
- Ethereum supports contract technologies that enable external entities to interact with the Blockchain ledger and enforce the rules over the data stored on the ledger.
- Moreover, Ethereum has extensive documentation, large community support, and development tools available.
- In addition, Ethereum Blockchain seems to be the most used public Blockchain in many studies, thus, it is easier to improve, extend and identify current encountered challenges that are outlined by the existing studies.

Ethereum Blockchain utilizes the Ethereum Virtual Machine (EVM) to compile and run contracts. Contracts are created using a solidity programming language which is the language made solely for developing Ethereum contracts. The Ethereum development tools and libraries utilized in the development and implementation of the proposed contract are illustrated in Figure 5.1.

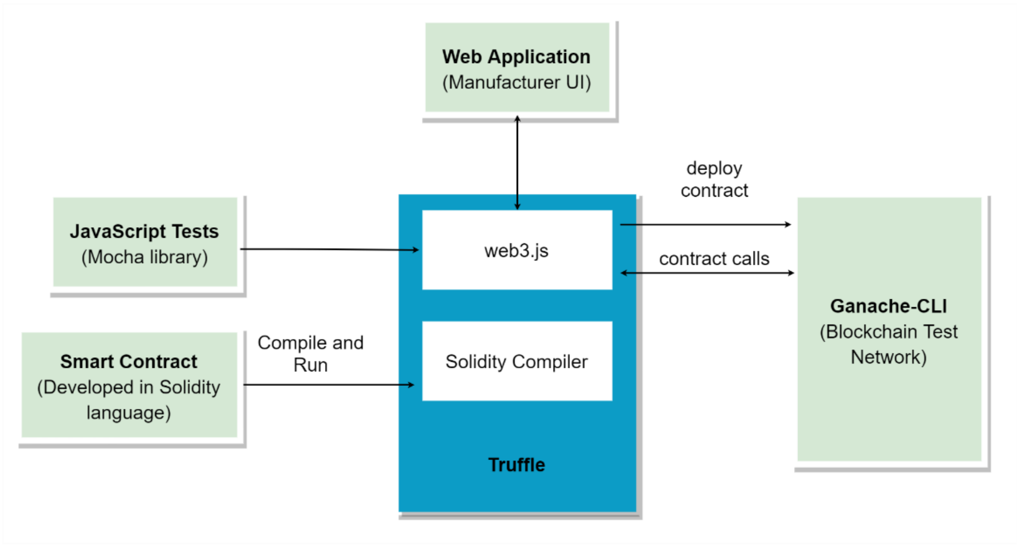

Web Application (Manufacturer UI)
JavaScript Tests (Mocha library)
web3.js
deploy contract
contract calls
Ganache-CLI (Blockchain Test Network)
Smart Contract (Developed in Solidity language)
Compile and Run
Solidity Compiler
Truffle

Figure 5.1 Development Tools and Libraries.

Ganache-CLI

This is a local Blockchain simulator that features a graphical user interface to simulate the networks and provide contract testing without the need to set up real Ethereum networks. It consists of fake Ethereum addresses to be used for testing purposes.

Truffle

Truffle is a development environment that integrates the compilation, testing, and deployment of Ethereum contracts. It is used to build and deploy decentralized applications for testing purposes.

Web3.js

Web3.js is a JavaScript library that implements the JSON-RPC protocol. It is used in the web application to connect and interact with the Blockchain network.

These tools were used to create, test, and deploy both FUS and manufacturer contracts. Unit tests were performed to validate the correct execution of contract functions and also to measure the costs of execution of the function in the network.

5.2 Data Storage

The proposed contracts have several storage and memory variables. The memory variable is a temporary place to store Blockchain data which gets erased between external function calls. Storage holds persistent data and is visible in all contract functions. Important data is stored in storage variables and guarantees its preservation.

The proposed contracts utilize two data structures namely arrays and mappings. An array can be of a fixed or dynamic size. The only downside of the arrays is the gas consumption. The array can consume too much gas, when it is big and searching for a specific value, which requires iterating the entire array,

It becomes more costly and may exceed the gas limit leading the contract to terminate the operation. Mappings allow storing key-value and using the key to access the data. It is costly to loop through the entire entries to find the desired data while with the arrays. Both mapping and array can be used together as the optimal solution to access the data to reduce costs. For example, the mapping keys can be stored in a separate array while the actual data is stored in the mapping. This guarantees that data accessibility is not lost but it comes at the cost of more storage.

5.2.1 Contract State Variables

This section explains the purpose of the storage variable used in both contracts. Table 5.2 lists and summarizes each storage variable presented in both contracts.

Table 5.2 Smart Contracts Variables

Smart Contract	Variables	Description
FUS	FUS_IDs	Holds the wallet address of the firmware update service at the time of deployment.
	FUS_name	Holds the FUS owner's name during the time of deployment.
	devices → (dev_id → Device)	Mapping of device ids to their respective device information which is represented by the struct.
	number_of_devices	The number of registered LoRa devices.
Manufacturer	manufacturer_id	Holds the wallet address of the manufacturer at the time of deployment.
	FUS_name	Holds the manufacturer's name during the time of deployment.
	Metadata_list → (model → metadata)	Mapping of models to their respective metadata
	Model_list []	List of device models of LoRa devices.

The FUS_ID, manufacturer_id, FUS_name, and Manufacturer state variables serve to hold the contract deployer's details. This is so important since it is necessary to know who owns the contract. Certain data is only accessible by the deployer of the contract. The FUS_ID and manufacturer_id variables hold the Blockchain addresses which are later used to authorize and give access to the data. The devices mapping data structure store all the registered LoRa devices that need to be updated via Blockchain. The dev_id is used as a key to map the LoRa end device.

```
struct Device{
    string appID;
    string devID;
    string s_address;
    string w_address;
    string model;
    string version;
    string status;
    bool exist;
}
```

Figure 5.2 Figure: Device Structure.

This is the same device id that was provided in the LoRaWAN application console during the device registration. The dev_id maps to the Device struct which is a solidity structure that enable us to create custom data type. The Device Struct is presented in Figure 5.2. The device structure consists of the LoRa device information which includes: the application id, the device id, the device manufacturer’s contract address, the manufacturer’s wallet address, the model, the currently installed version, the update status, and the ‘exist’ variable. The ’exist’ variable helps to avoid adding the new device that already exists in Blockchain. The Manufacturer’s contract has a metadata_list mapping which maps a model with the respective latest metadata. The metadata is also represented in structure as shown in Figure 5.3. The structure is made up of firmware version and metadata variable which represents the actual metadata. The model list variable is an array that keeps track of all device models that have firmware metadata. It is useful when checking if the device model exists before checking its new firmware update availability.

```
struct FirmwareMetadata{
    string f_model;
    string f_version;
    string f_metadata;
}
```

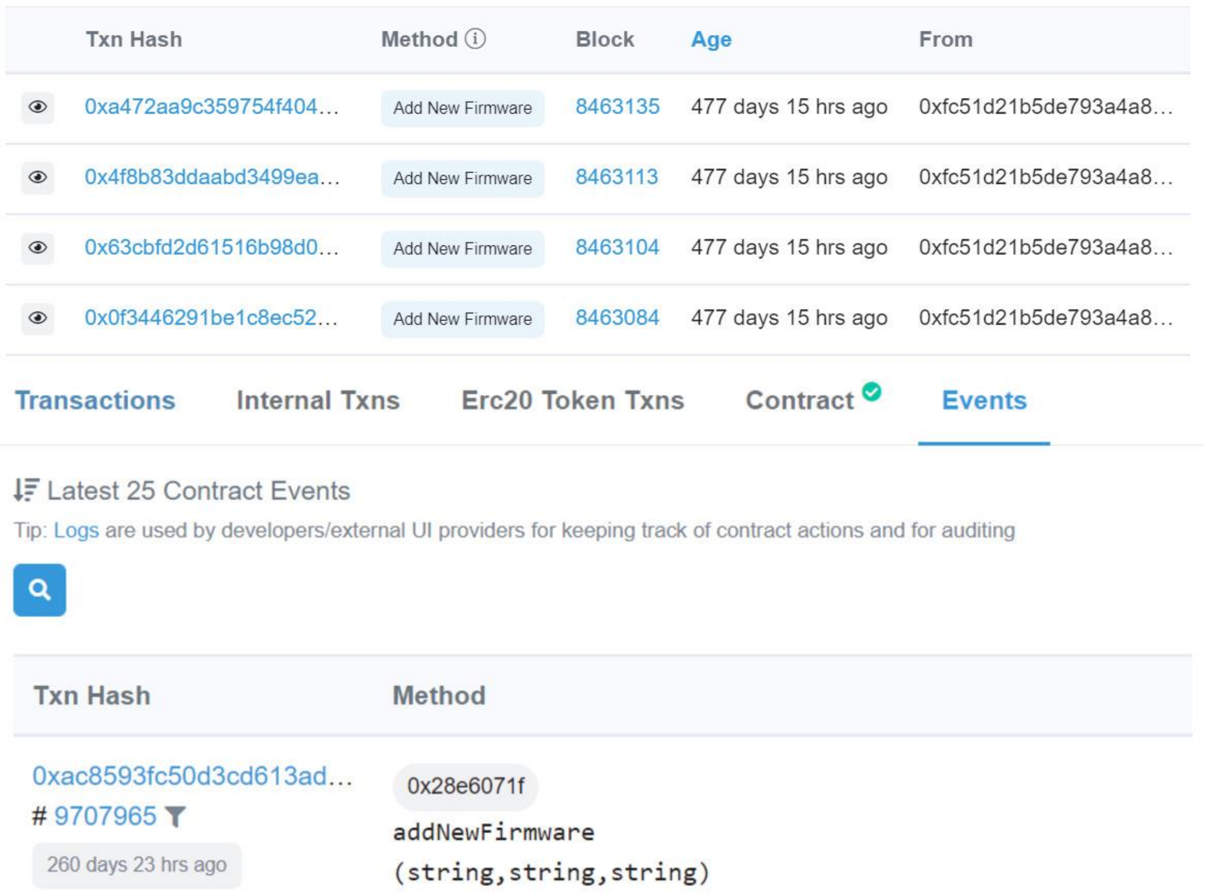
Figure 5.3 Metadata Solidity Structure

5.2.2 Methods and Functionalities

This section describes important implementations of the contract algorithms proposed in 4.7. and also shows the important lines of implantation code of the functions. These functions use the storage variable discussed in 5.2.1. Appendix A shows the addNewFirmware() function that adds new metadata to the Blockchain. The function takes the model, the new version, and the metadata of the firmware. The device's model was checked to ascertain that it exists in the Blockchain and if not, it gets added to the model_list array defined in Section 5.2.1.

The purpose of adding the model to the model_list array is to check the provided model at a time when the isUpdateAvailable() function is called as shown in Appendix B. When isUpdateAvailable() is called the model is checked to be sure it exists. It can only exist after the addNewFirmware() function is called. The addNewFirmware() function creates the in-memory metadata structure which gets added to the list of metadata. When the new metadata is added, the LoRa devices or FUS need to know they should be updated. The function emits the event to broadcast the new arrival of metadata. Moreover, the function ensures that is only executed by the manufacturer of the devices on the network. This is achieved by using the required statement which requires that the entity which calls the function should be the one who deployed the contract that is the address of the manufacturer (manufacturer_id).

Figure 5.4 demonstrates the transaction made by the manufacturer when the new firmware metadata is uploaded and shows the events that get triggered when the new metadata is uploaded. The Blockchain event can be disabled or enabled in the configuration file. This is shown with the key value of 'auto-updates' in Figure 5.5. When the value is set to 'True', the event is enabled, and the FUS will start a Blockchain event thread shown in Figure 5.6 in line 63 that listens for new firmware updates. This illustrates how the auto-updates take place. Figure 5.6 also demonstrates that if the FUS receives the new metadata, it gets published via the MQTT topic (as shown in line 58) that will handle the metadata received and start the firmware update process.



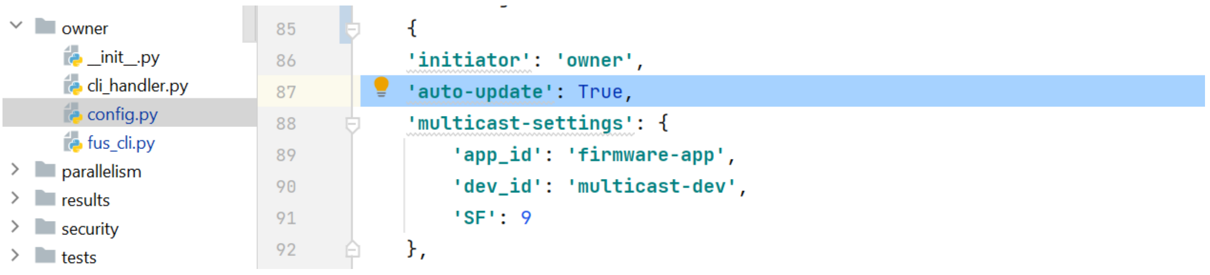
	Txn Hash	Method	Block	Age	From
👁	0xa472aa9c359754f404…	Add New Firmware	8463135	477 days 15 hrs ago	0xfc51d21b5de793a4a8…
👁	0x4f8b83ddaabd3499ea…	Add New Firmware	8463113	477 days 15 hrs ago	0xfc51d21b5de793a4a8…
👁	0x63cbfd2d61516b98d0…	Add New Firmware	8463104	477 days 15 hrs ago	0xfc51d21b5de793a4a8…
👁	0x0f3446291be1c8ec52…	Add New Firmware	8463084	477 days 15 hrs ago	0xfc51d21b5de793a4a8…

Latest 25 Contract Events

Tip: Logs are used by developers/external UI providers for keeping track of contract actions and for auditing

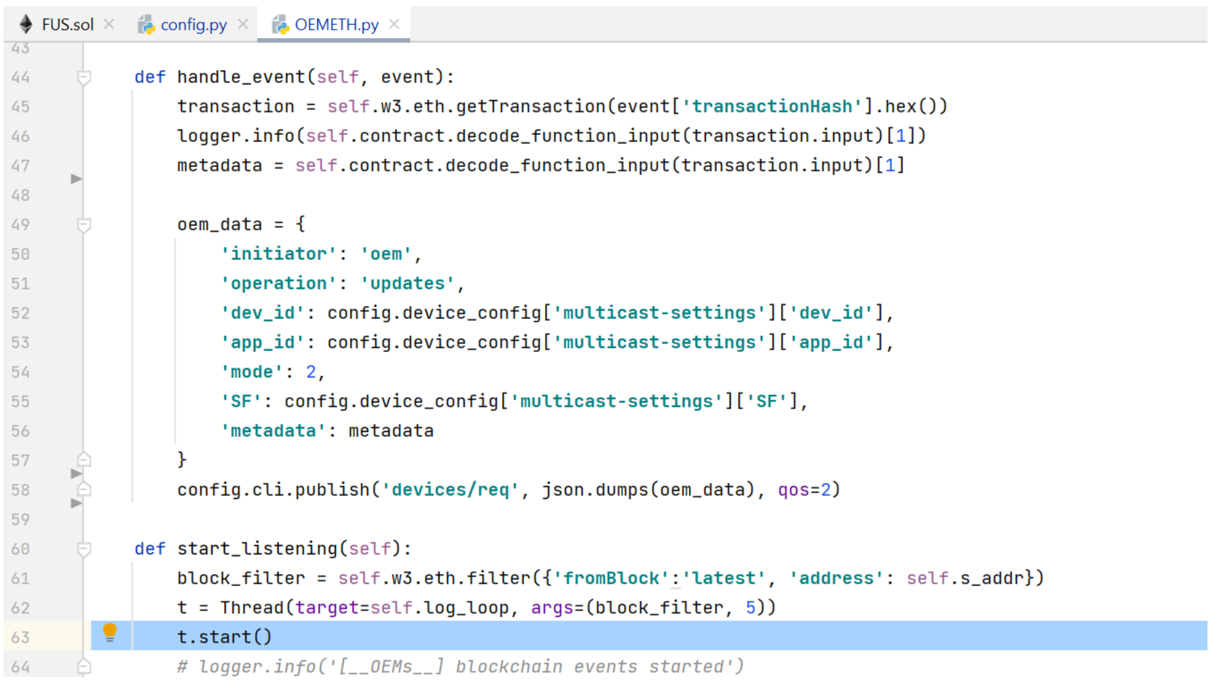
Txn Hash	Method
0xac8593fc50d3cd613ad… # 9707965 260 days 23 hrs ago	0x28e6071f addNewFirmware (string,string,string)

Figure 5.4 Blockchain Transactions and Event for Firmware Metadata Upload.



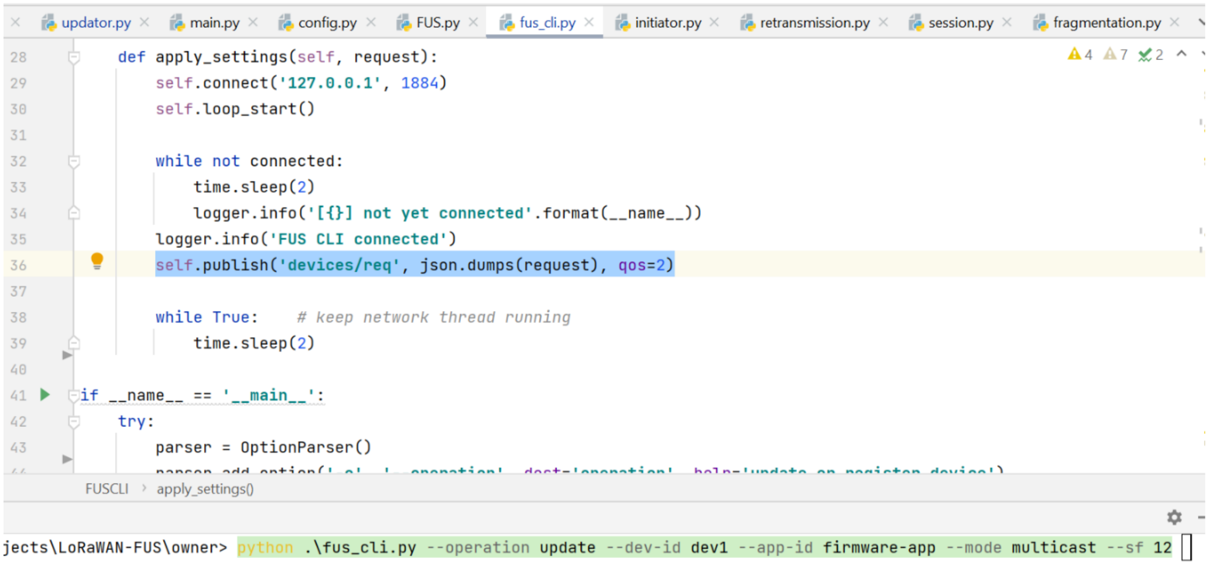
```
{
    'initiator': 'owner',
    'auto-update': True,
    'multicast-settings': {
        'app_id': 'firmware-app',
        'dev_id': 'multicast-dev',
        'SF': 9
    },
```

Figure 5.5 Enable and Disable Auto-Updates.



```
    def handle_event(self, event):
        transaction = self.w3.eth.getTransaction(event['transactionHash'].hex())
        logger.info(self.contract.decode_function_input(transaction.input)[1])
        metadata = self.contract.decode_function_input(transaction.input)[1]

        oem_data = {
            'initiator': 'oem',
            'operation': 'updates',
            'dev_id': config.device_config['multicast-settings']['dev_id'],
            'app_id': config.device_config['multicast-settings']['app_id'],
            'mode': 2,
            'SF': config.device_config['multicast-settings']['SF'],
            'metadata': metadata
        }
        config.cli.publish('devices/req', json.dumps(oem_data), qos=2)

    def start_listening(self):
        block_filter = self.w3.eth.filter({'fromBlock':'latest', 'address': self.s_addr})
        t = Thread(target=self.log_loop, args=(block_filter, 5))
        t.start()
        # logger.info('[__OEMs__] blockchain events started')
```

Figure 5.6 Starting the Blockchain Event.



```
    def apply_settings(self, request):
        self.connect('127.0.0.1', 1884)
        self.loop_start()

        while not connected:
            time.sleep(2)
            logger.info('[{}] not yet connected'.format(__name__))
        logger.info('FUS CLI connected')
        self.publish('devices/req', json.dumps(request), qos=2)

        while True:    # keep network thread running
            time.sleep(2)

if __name__ == '__main__':
    try:
        parser = OptionParser()
        parser.add_option('-o', '--operation', dest='operation', help='update or register device')
```

```
jects\LoRaWAN-FUS\owner> python .\fus_cli.py --operation update --dev-id dev1 --app-id firmware-app --mode multicast --sf 12
```

Figure 5.7 Updates the Device via CLI.

When the value of 'auto-updates' in the configuration file is set to 'False', it means the firmware update process could be only started by the FUS owner via the CLI script. Figure 5.7 shows the snapshot of the fus_cli.py script implementation utilized by the device owner to initiate the firmware update process. When the owner initiates the firmware updates, the FUS call is UpdateAvailable() function shown in Appendix A which checks if there is any new firmware available by using the model and the current provided version. The current version of the device is compared with the latest version available on the Blockchain network. If the provided version of the model is less than the latest version the function returns true or else false if the provided version is equal.

The fus_cli.py script is not only limited to initiating the firmware update process, it can also be used to manage the end devices registered in the Blockchain. This means the owner can connect to the Blockchain via the script to get metadata, register, delete, update device status, update device state, and get device information operations. The implementation of these operations is illustrated in Appendixes C, D, E, F, G, and H respectively.

The Blockchain functions such as registering, deleting, and updating that make a transaction to Blockchain must be called by the owner of the contract otherwise, if a different entity calls them, they would not be executed successfully. The registerDev() registers the LoRa device information into the Blockchain. The function takes the information as an argument and represents it in the Device structure presented in Figure 5.2. The number of devices variable gets updated since there is a new device being added. The created device gets added to the mapping of the device. Once the device is created, it may be deleted when it is no longer needed. The deleteDev() function can be called to delete it. The function takes the dev_ID as an argument that serves as a key to the device mapping. Once the device is deleted in the mapping the number_of_devices storage variable is decremented.

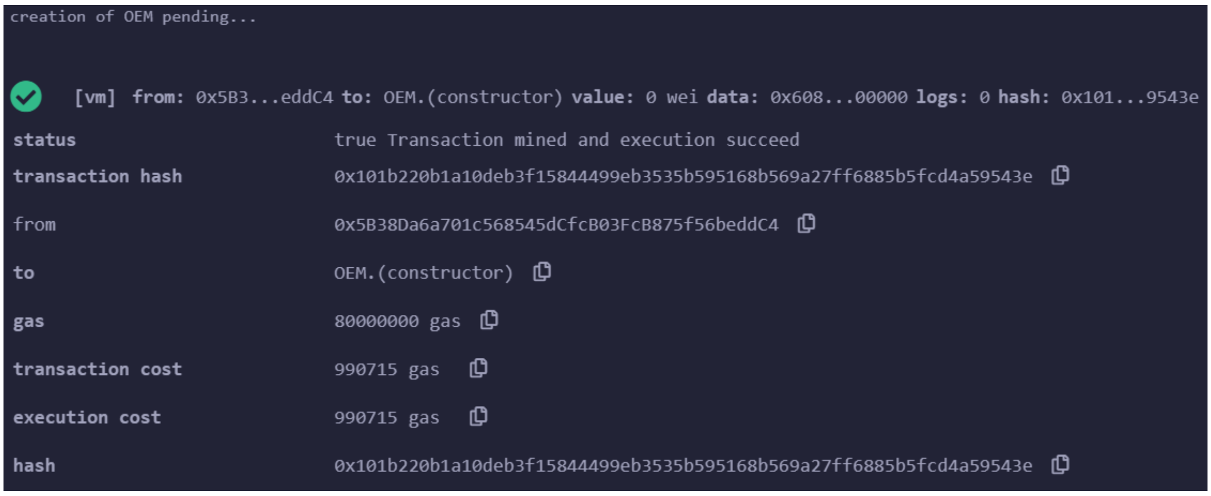
The updating functions, the updateDeviceInfo(), and updateDeviceStatus() use the required statement to authorize only the device owner to manipulate the device. The updateDeviceInfo() accepts the information required to update the device structure as arguments presented in Figure 5.2. The devices mapping is then utilized to retrieve the device using the devID then the existing device information is updated with the input arguments. The updateDeviceStatus takes the device ID and the status. The device ID is used to retrieve the device on devices mapping to update the status. The getDevInfo() function only takes the device ID and returns the corresponding device's information. The other get operations such as getDeviceStatus also take the device ID and return the corresponding device's status. The retrieveMetadata() and getDevicesByModel() both take the model as arguments. retrieveMetadata() returns firmware metadata represented as key-value pair shown in Figure 5.32 and getDevicesByModel() returns the list of device IDs that can be used to retrieve the set of devices.

5.3 Testing and Validation of the Smart Contract

In this section, we test the proposed smart contracts operations and present the log results. In our testing, the functions are tested for their functionality as well as the access control. Each function is executed by an entity with the right access to execute it, meaning there is access control posed by function data on the Blockchain during the firmware update. In addition, events and their logs are checked to ensure the function executes as intended. Each smart contract has an owner. A smart contract identifies its owner with an owner's address. This owner's address was captured during a smart contract deployment. Two contracts mean that two entities own the contracts namely: the device manufacturer and owner. Their addresses are shown in the table below.

Entity	Addresses
Manufacturer	0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2
Device Owner	0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
Attacker	0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db

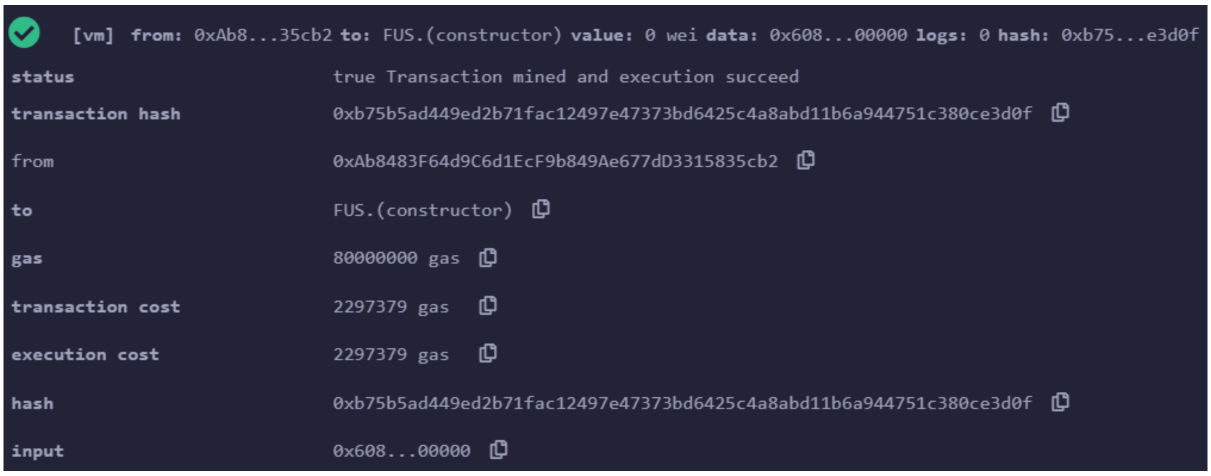
The functions are tested in Remix IDE and their results are shown in the snapshots.



```
creation of OEM pending...

[vm]  from: 0x5B3...eddC4 to: OEM.(constructor) value: 0 wei data: 0x608...00000 logs: 0 hash: 0x101...9543e
status                      true Transaction mined and execution succeed
transaction hash            0x101b220b1a10deb3f15844499eb3535b595168b569a27ff6885b5fcd4a59543e
from                        0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                          OEM.(constructor)
gas                         80000000 gas
transaction cost            990715 gas
execution cost              990715 gas
hash                        0x101b220b1a10deb3f15844499eb3535b595168b569a27ff6885b5fcd4a59543e
```

Figure 5.8 Manufacturer Contract Deployment Logs

During the deployment of both smart contracts, the device manufacturer and owners are identified with the addresses 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 and 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2 respectively.

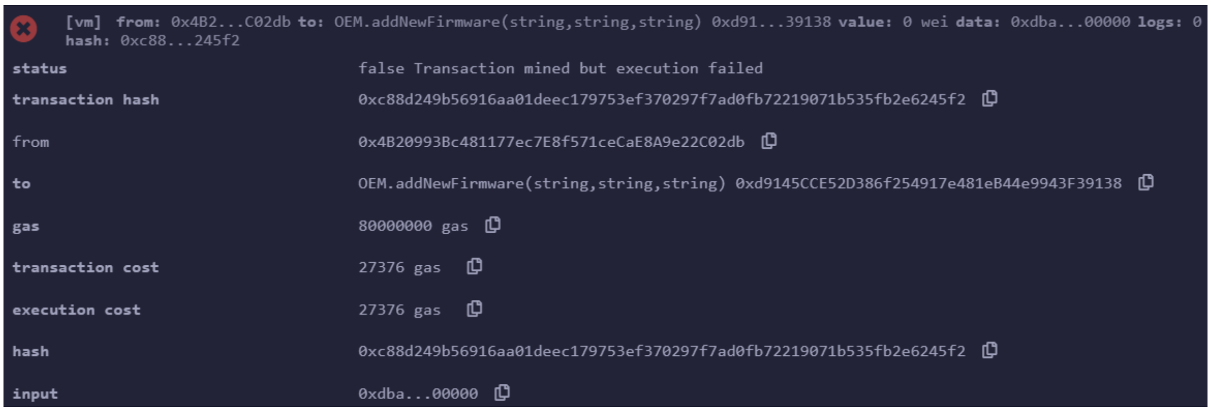


```
[vm] from: 0xAb8...35cb2 to: FUS.(constructor) value: 0 wei data: 0x608...00000 logs: 0 hash: 0xb75...e3d0f
status                        true Transaction mined and execution succeed
transaction hash              0xb75b5ad449ed2b71fac12497e47373bd6425c4a8abd11b6a944751c380ce3d0f
from                          0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2
to                            FUS.(constructor)
gas                           80000000 gas
transaction cost              2297379 gas
execution cost                2297379 gas
hash                          0xb75b5ad449ed2b71fac12497e47373bd6425c4a8abd11b6a944751c380ce3d0f
input                         0x608...00000
```

Figure 5.9 Device Owner Contract Deployment Logs

Figure 5.8 and Figure 5.9 show successful deployment transactions of both smart contracts. It is at this point where Algorithm 1 and Algorithm 5 execute. Both contracts' constructors assign the aforementioned address to the Blockchain storage.

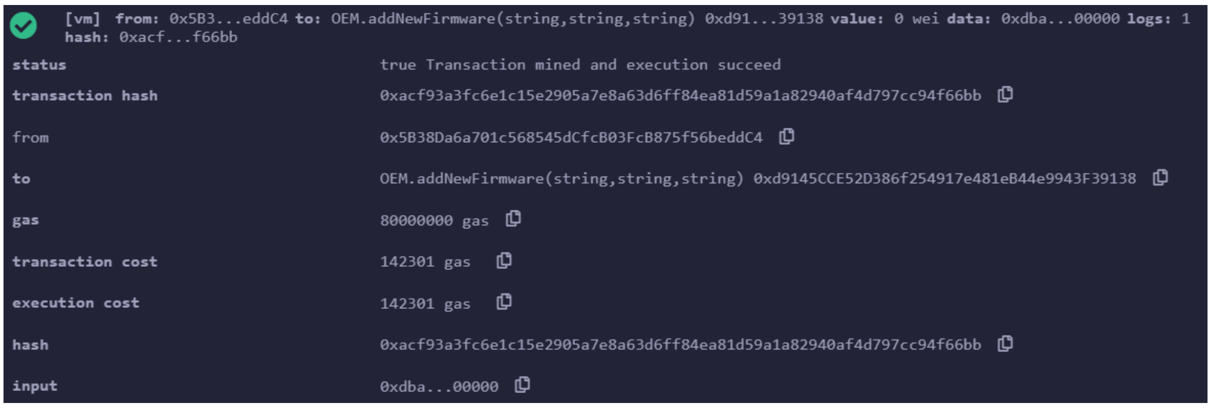
Adding metadata to the Blockchain network requires the 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 address, which was the address that deployed the device manufacturer's contract.



```
[vm] from: 0x4B2...C02db to: OEM.addNewFirmware(string,string,string) 0xd91...39138 value: 0 wei data: 0xdba...00000 logs: 0
hash: 0xc88...245f2
status                        false Transaction mined but execution failed
transaction hash              0xc88d249b56916aa01deec179753ef370297f7ad0fb72219071b535fb2e6245f2
from                          0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db
to                            OEM.addNewFirmware(string,string,string) 0xd9145CCE52D386f254917e481eB44e9943F39138
gas                           80000000 gas
transaction cost              27376 gas
execution cost                27376 gas
hash                          0xc88d249b56916aa01deec179753ef370297f7ad0fb72219071b535fb2e6245f2
input                         0xdba...00000
```

Figure 5.10 Add New Metadata Transaction Failure Logs.

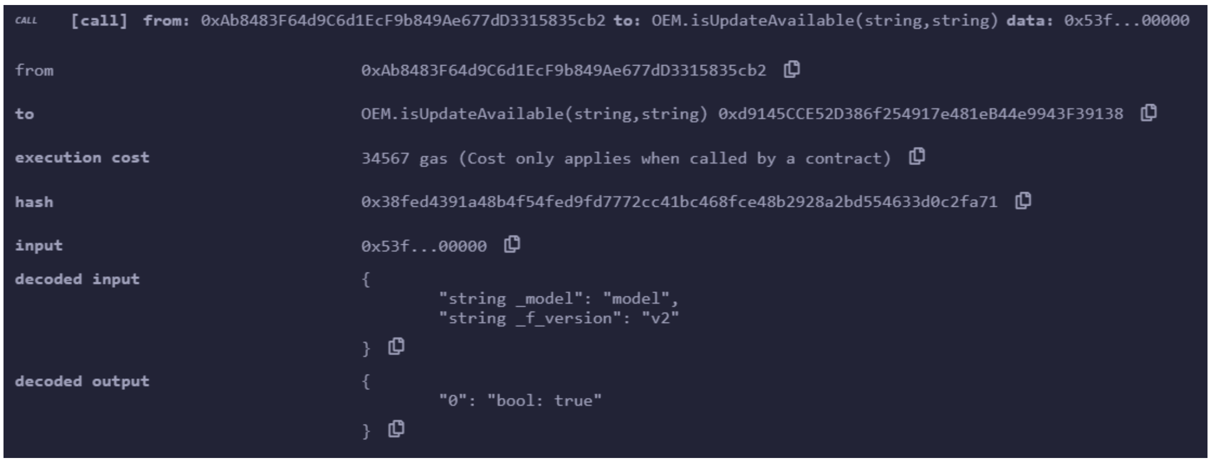
Figure 5.10 shows that if a different address tries to add the metadata to the Blockchain the transaction will be unsuccessful. This is illustrated with an attacker's address 0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db which invokes the addMetadata operation shown in Algorithm 2. In addition, the event that emits upon the upload of the metadata will not be called.



```
[vm]  from: 0x5B3...eddC4 to: OEM.addNewFirmware(string,string,string) 0xd91...39138 value: 0 wei data: 0xdba...00000 logs: 1
hash: 0xacf...f66bb
status                     true Transaction mined and execution succeed
transaction hash           0xacf93a3fc6e1c15e2905a7e8a63d6ff84ea81d59a1a82940af4d797cc94f66bb
from                       0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                         OEM.addNewFirmware(string,string,string) 0xd9145CCE52D386f254917e481eB44e9943F39138
gas                        80000000 gas
transaction cost           142301 gas
execution cost             142301 gas
hash                       0xacf93a3fc6e1c15e2905a7e8a63d6ff84ea81d59a1a82940af4d797cc94f66bb
input                      0xdba...00000
```

Figure 5.11 Add New Metadata Transaction Success Logs

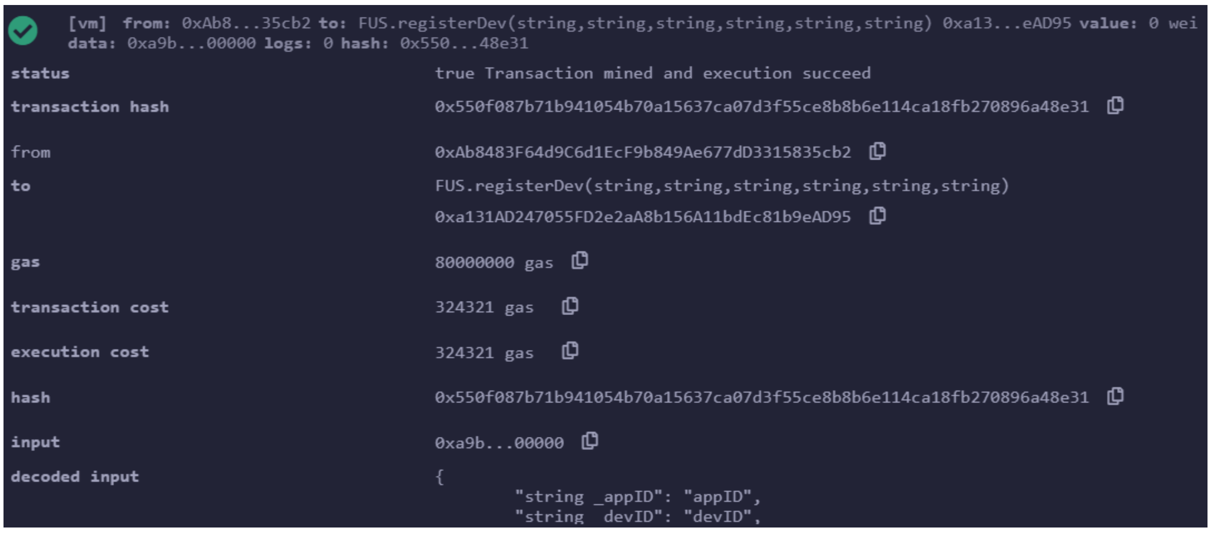
A successful transaction of adding metadata is only possible with the manufacturer's address 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 as shown in Figure 5.11 and this will result in the successful emission of upload metadata event. The question is if the impersonation is possible through this address. This address corresponds to a private key, and that private key is owned and held secret by a single entity. This means spoofing may be possible when someone knows another person's private key. There is no way an attacker could successfully impersonate a Blockchain address without knowing their private key and the private key is irreversible. Unlike, in traditional networks where one can do IP address spoofing when one creates data with a false source IP address to impersonate another entity. In this case, the public key is not changeable and is only produced through the private key. It is not easy to manufacture a private key from a public key. The attack is possible when an attacker changes the data itself or the address. However, when that data gets to the Blockchain, it will not be added since the signature and the address will not correspond to the expected address. This is also demonstrated in Figure 5.10.



```
CALL  [call]  from: 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2 to: OEM.isUpdateAvailable(string,string) data: 0x53f...00000
from                       0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2
to                         OEM.isUpdateAvailable(string,string) 0xd9145CCE52D386f254917e481eB44e9943F39138
execution cost             34567 gas (Cost only applies when called by a contract)
hash                       0x38fed4391a48b4f54fed9fd7772cc41bc468fce48b2928a2bd554633d0c2fa71
input                      0x53f...00000
decoded input              {
                                   "string _model": "model",
                                   "string _f_version": "v2"
                           }
decoded output             {
                                   "0": "bool: true"
                           }
```

Figure 5.12 Check New Firmware Update Logs

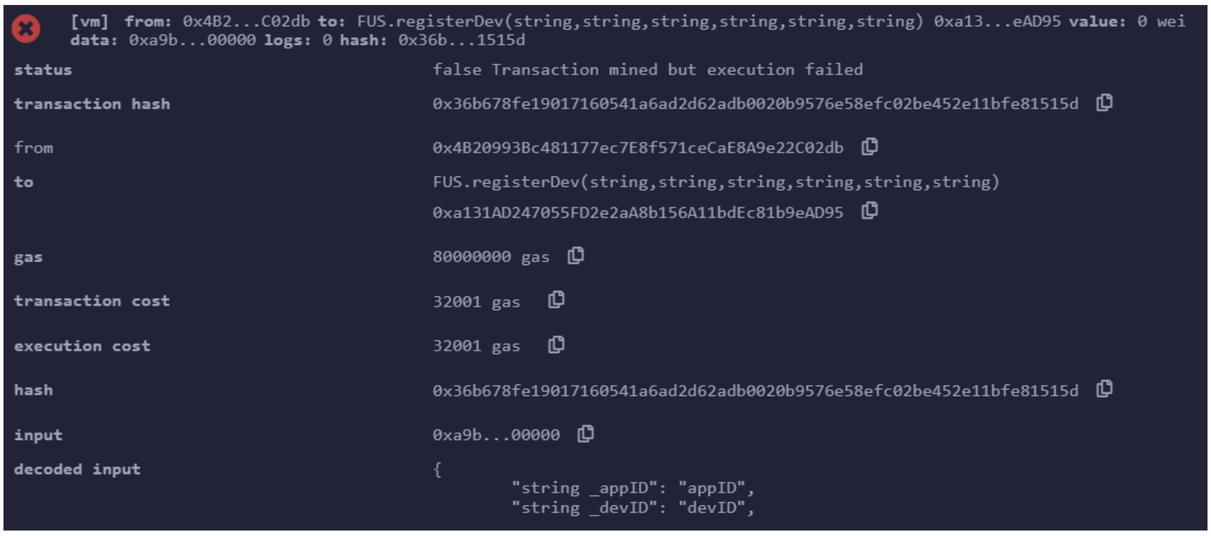
After the metadata is uploaded then the isUpdateAvailable operation can be called. This is operation is tested with an address 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2. Note, this operation does not require any special access meaning it does not need to be called by only the manufacturer but any entity that wants to check for firmware updates can invoke the operation.



```
[vm] from: 0xAb8...35cb2 to: FUS.registerDev(string,string,string,string,string,string) 0xa13...eAD95 value: 0 wei
data: 0xa9b...00000 logs: 0 hash: 0x550...48e31
status                  true Transaction mined and execution succeed
transaction hash        0x550f087b71b941054b70a15637ca07d3f55ce8b8b6e114ca18fb270896a48e31

from                    0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2
to                      FUS.registerDev(string,string,string,string,string,string)
                        0xa131AD247055FD2e2aA8b156A11bdEc81b9eAD95

gas                     80000000 gas

transaction cost        324321 gas

execution cost          324321 gas

hash                    0x550f087b71b941054b70a15637ca07d3f55ce8b8b6e114ca18fb270896a48e31

input                   0xa9b...00000
decoded input           {
                                "string _appID": "appID",
                                "string _devID": "devID",
```

Figure 5.13 Register Device Transaction Pass Logs

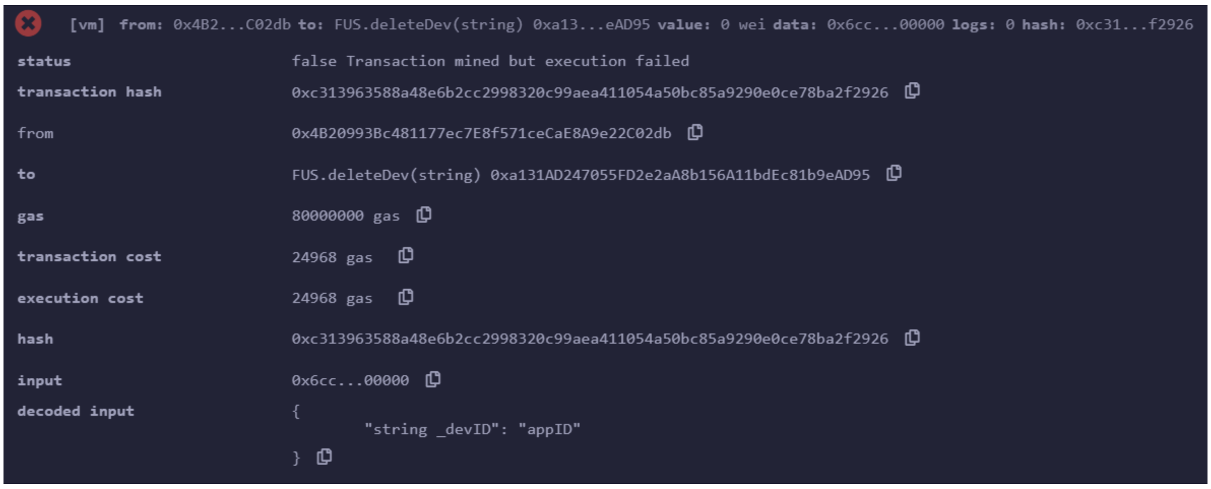
Device registration to the Blockchain network is only permitted to be done by the address 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2. Figure 5.13 shows a successful registration of the low-powered device to the network where the owner's address was used to execute the registerDevice() operation.



```
[vm] from: 0x4B2...C02db to: FUS.registerDev(string,string,string,string,string,string) 0xa13...eAD95 value: 0 wei
data: 0xa9b...00000 logs: 0 hash: 0x36b...1515d
status                  false Transaction mined but execution failed
transaction hash        0x36b678fe19017160541a6ad2d62adb0020b9576e58efc02be452e11bfe81515d

from                    0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db
to                      FUS.registerDev(string,string,string,string,string,string)
                        0xa131AD247055FD2e2aA8b156A11bdEc81b9eAD95

gas                     80000000 gas

transaction cost        32001 gas

execution cost          32001 gas

hash                    0x36b678fe19017160541a6ad2d62adb0020b9576e58efc02be452e11bfe81515d

input                   0xa9b...00000
decoded input           {
                                "string _appID": "appID",
                                "string _devID": "devID",
```

Figure 5.14 Register Device Transaction Failure Logs

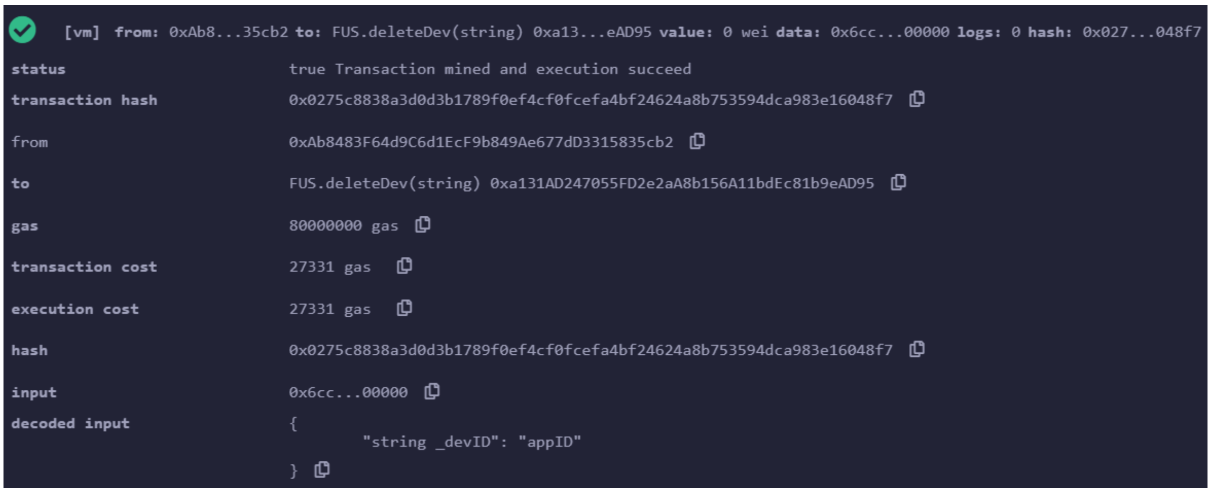
When testing this operation with the 0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db address, the operation execution fails because only the owner's address is permitted to execute this operation.



```
[vm]  from: 0x4B2...C02db to: FUS.deleteDev(string) 0xa13...eAD95 value: 0 wei data: 0x6cc...00000 logs: 0 hash: 0xc31...f2926
status                      false Transaction mined but execution failed
transaction hash            0xc313963588a48e6b2cc2998320c99aea411054a50bc85a9290e0ce78ba2f2926

from                        0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db

to                          FUS.deleteDev(string) 0xa131AD247055FD2e2aA8b156A11bdEc81b9eAD95

gas                         80000000 gas

transaction cost            24968 gas

execution cost              24968 gas

hash                        0xc313963588a48e6b2cc2998320c99aea411054a50bc85a9290e0ce78ba2f2926

input                       0x6cc...00000
decoded input               {
                                    "string _devID": "appID"
                            }
```

Figure 5.15 Delete Device Transaction Failure Logs

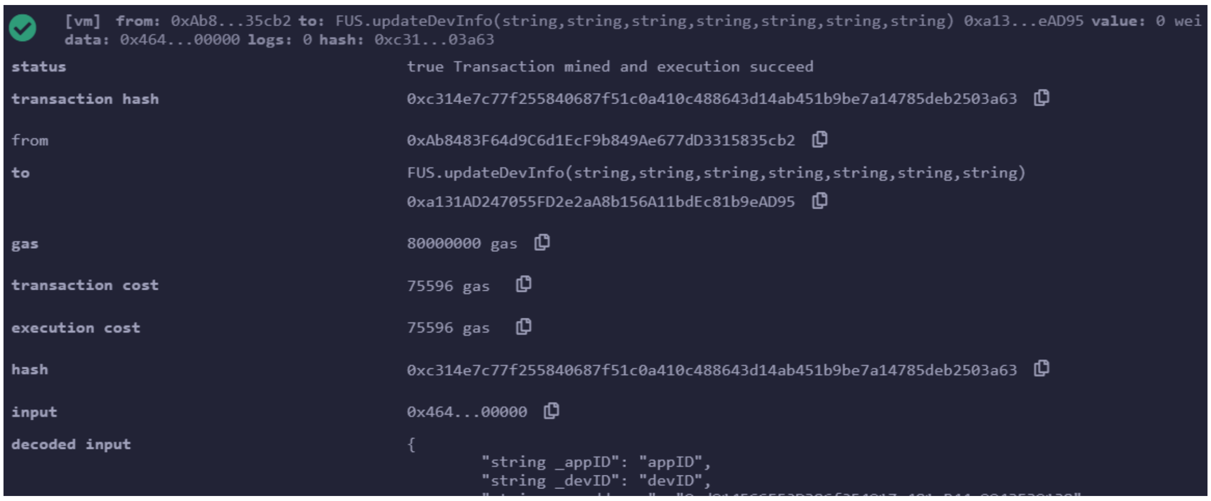
Deleting a device is one of the crucial operations. To test this operation to see whether it behaves as intended and only permits the device owner to delete the device, we utilized an address 0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db. If this address calls the operation, the transaction will be successfully mined but results in an unsuccessful execution since it will fail due to a calling address.



```
[vm]  from: 0xAb8...35cb2 to: FUS.deleteDev(string) 0xa13...eAD95 value: 0 wei data: 0x6cc...00000 logs: 0 hash: 0x027...048f7
status                      true Transaction mined and execution succeed
transaction hash            0x0275c8838a3d0d3b1789f0ef4cf0fcefa4bf24624a8b753594dca983e16048f7

from                        0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2

to                          FUS.deleteDev(string) 0xa131AD247055FD2e2aA8b156A11bdEc81b9eAD95

gas                         80000000 gas

transaction cost            27331 gas

execution cost              27331 gas

hash                        0x0275c8838a3d0d3b1789f0ef4cf0fcefa4bf24624a8b753594dca983e16048f7

input                       0x6cc...00000
decoded input               {
                                    "string _devID": "appID"
                            }
```

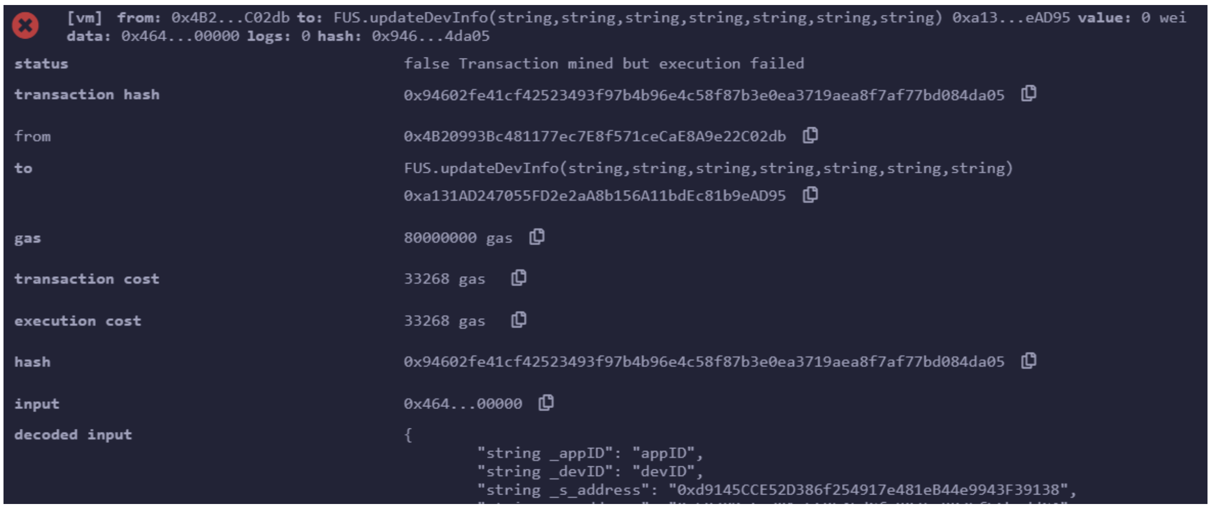
Figure 5.16 Delete Device Transaction Pass Logs

Figure 5.16 demonstrates that only the address 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2 can successfully execute the transaction.



```
[vm] from: 0xAb8...35cb2 to: FUS.updateDevInfo(string,string,string,string,string,string,string) 0xa13...eAD95 value: 0 wei
data: 0x464...00000 logs: 0 hash: 0xc31...03a63
status                     true Transaction mined and execution succeed
transaction hash           0xc314e7c77f255840687f51c0a410c488643d14ab451b9be7a14785deb2503a63
from                       0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2
to                         FUS.updateDevInfo(string,string,string,string,string,string,string)
                           0xa131AD247055FD2e2aA8b156A11bdEc81b9eAD95
gas                        80000000 gas
transaction cost           75596 gas
execution cost             75596 gas
hash                       0xc314e7c77f255840687f51c0a410c488643d14ab451b9be7a14785deb2503a63
input                      0x464...00000
decoded input              {
                                   "string _appID": "appID",
                                   "string _devID": "devID",
```

Figure 5.17 Update Device Transaction Pass Logs



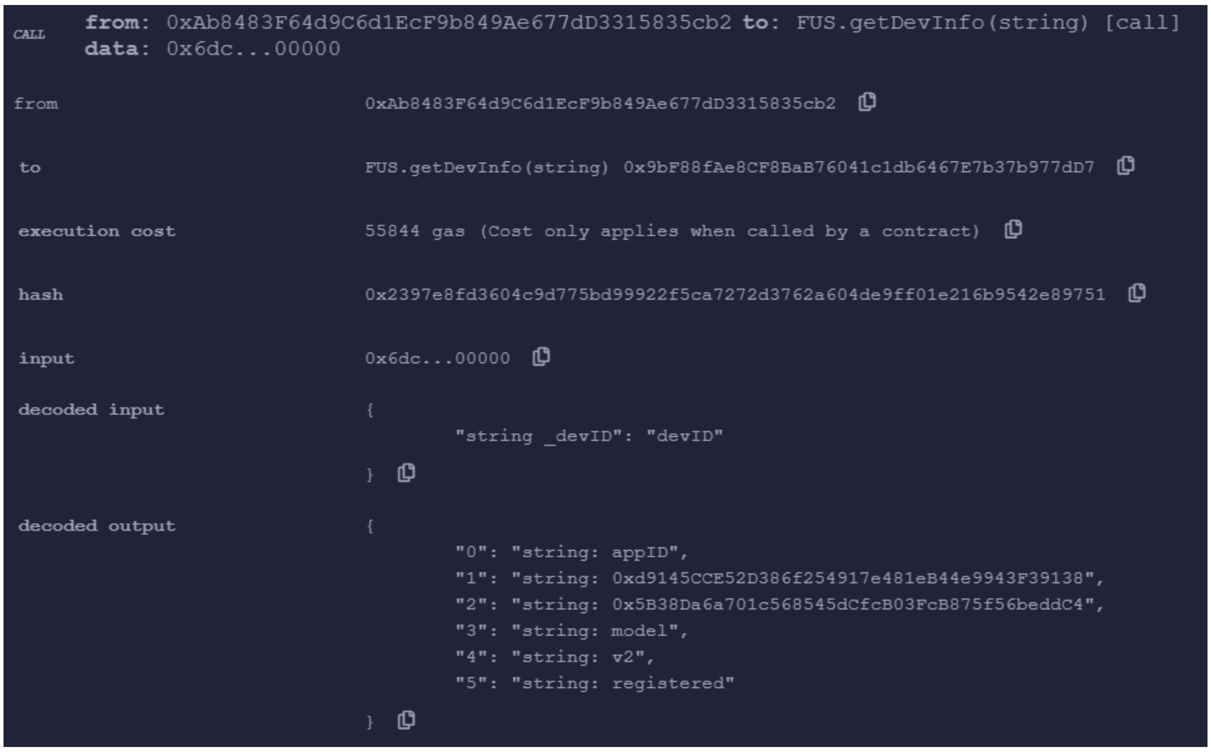
```
[vm] from: 0x4B2...C02db to: FUS.updateDevInfo(string,string,string,string,string,string,string) 0xa13...eAD95 value: 0 wei
data: 0x464...00000 logs: 0 hash: 0x946...4da05
status                     false Transaction mined but execution failed
transaction hash           0x94602fe41cf42523493f97b4b96e4c58f87b3e0ea3719aea8f7af77bd084da05
from                       0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db
to                         FUS.updateDevInfo(string,string,string,string,string,string,string)
                           0xa131AD247055FD2e2aA8b156A11bdEc81b9eAD95
gas                        80000000 gas
transaction cost           33268 gas
execution cost             33268 gas
hash                       0x94602fe41cf42523493f97b4b96e4c58f87b3e0ea3719aea8f7af77bd084da05
input                      0x464...00000
decoded input              {
                                   "string _appID": "appID",
                                   "string _devID": "devID",
                                   "string _s_address": "0xd9145CCE52D386f254917e481eB44e9943F39138",
```

Figure 5.18 Update Device Transaction Failure Logs

Updating the low-powered device information on the network requires the caller of the updateDevInfo() to be the owner of the contract.

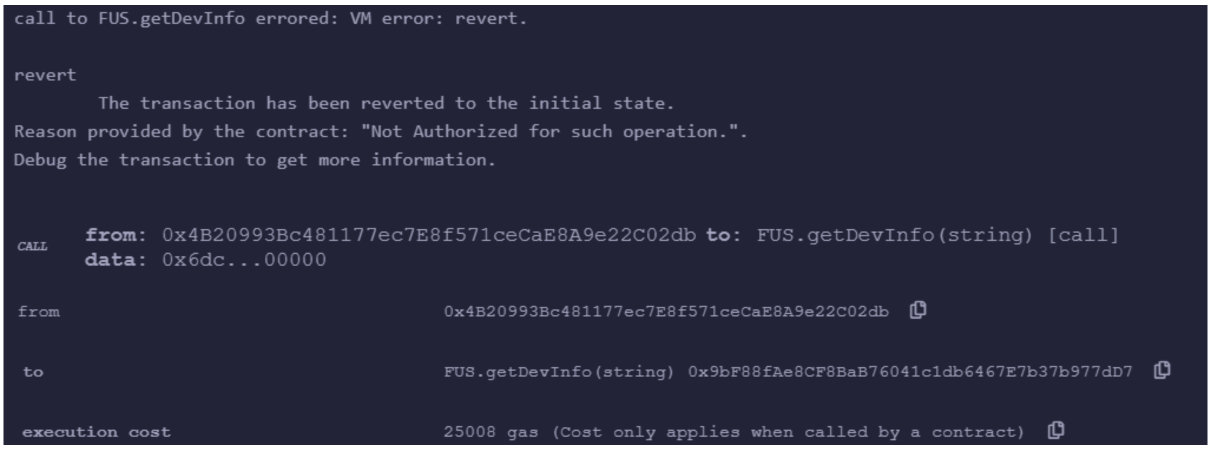
Figure 5.17 and Figure 5.18 shows the successful and unsuccessful execution of the updateDevInfo() transaction respectively where the address 0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db led to an unsuccessful transaction.

The contract operations that get device information are also restricted to being executed by the device owner only. All these operations were tested using two addresses: the device owner's address and the address 0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db representing an illegitimate entity calling the functions. The successful and unsuccessful test results of these functions including getDevInfo, getDevStatus are shown in Figure 5.19, Figure 5.20, Figure 5.21, and Figure 5.22 and respectively.



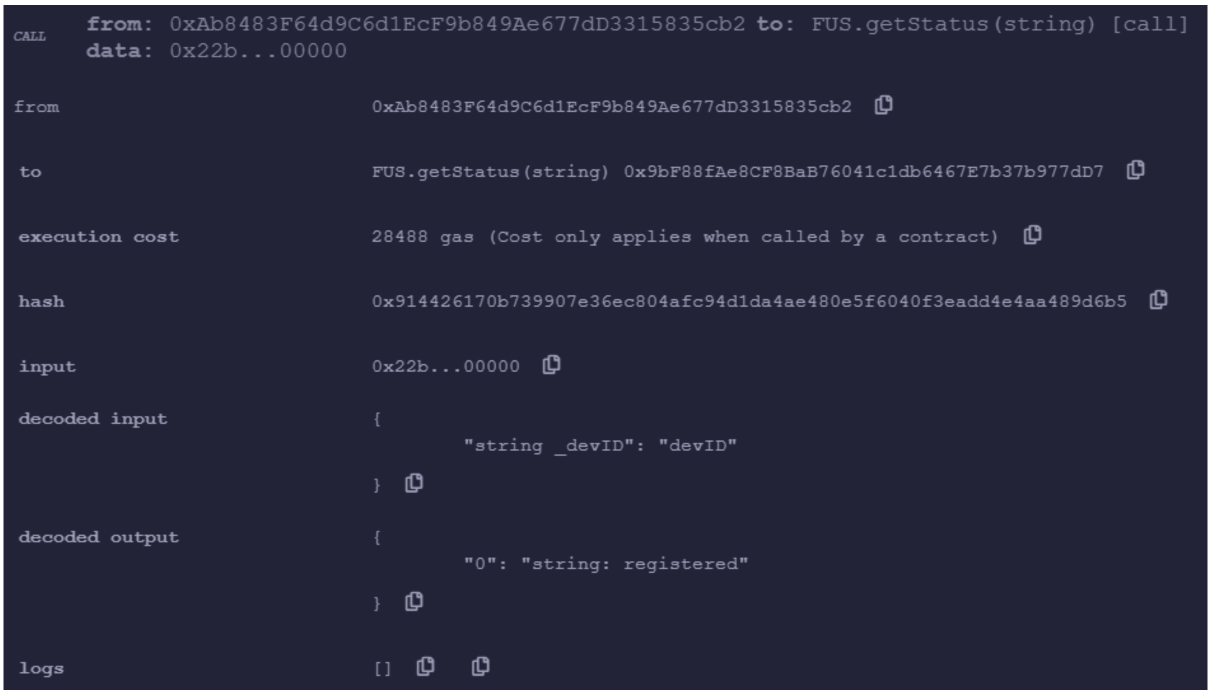
```
CALL   from: 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2 to: FUS.getDevInfo(string) [call]
       data: 0x6dc...00000

from                     0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2

to                       FUS.getDevInfo(string) 0x9bF88fAe8CF8BaB76041c1db6467E7b37b977dD7

execution cost           55844 gas (Cost only applies when called by a contract)

hash                     0x2397e8fd3604c9d775bd99922f5ca7272d3762a604de9ff01e216b9542e89751

input                    0x6dc...00000

decoded input            {
                                 "string _devID": "devID"
                         }

decoded output           {
                                 "0": "string: appID",
                                 "1": "string: 0xd9145CCE52D386f254917e481eB44e9943F39138",
                                 "2": "string: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4",
                                 "3": "string: model",
                                 "4": "string: v2",
                                 "5": "string: registered"
                         }
```

Figure 5.19 Get Device Information Transaction Pass Logs



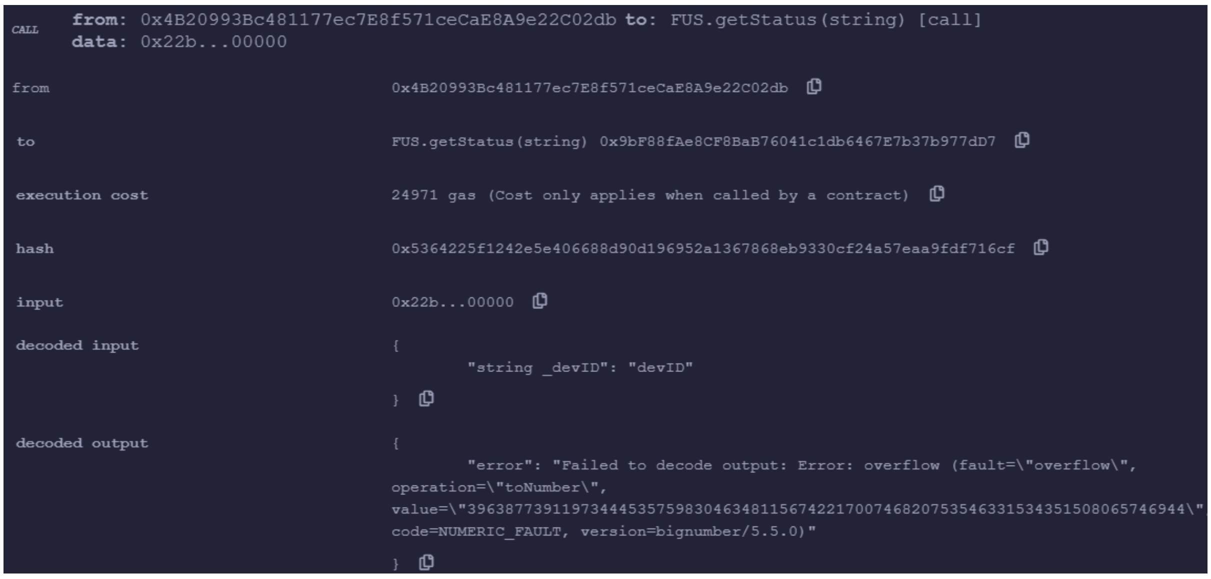
```
call to FUS.getDevInfo errored: VM error: revert.

revert
        The transaction has been reverted to the initial state.
Reason provided by the contract: "Not Authorized for such operation.".
Debug the transaction to get more information.


CALL   from: 0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db to: FUS.getDevInfo(string) [call]
       data: 0x6dc...00000

from                     0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db

to                       FUS.getDevInfo(string) 0x9bF88fAe8CF8BaB76041c1db6467E7b37b977dD7

execution cost           25008 gas (Cost only applies when called by a contract)
```

Figure 5.20 Get Device Information Transaction Failure Logs



```
CALL  from: 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2 to: FUS.getStatus(string) [call]
      data: 0x22b...00000

from               0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2

to                 FUS.getStatus(string) 0x9bF88fAe8CF8BaB76041c1db6467E7b37b977dD7

execution cost     28488 gas (Cost only applies when called by a contract)

hash               0x914426170b739907e36ec804afc94d1da4ae480e5f6040f3eadd4e4aa489d6b5

input              0x22b...00000

decoded input      {
                          "string _devID": "devID"
                   }

decoded output     {
                          "0": "string: registered"
                   }

logs               []
```

Figure 5.21 Get Device Status Transaction Pass Logs



```
CALL  from: 0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db to: FUS.getStatus(string) [call]
      data: 0x22b...00000

from               0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db

to                 FUS.getStatus(string) 0x9bF88fAe8CF8BaB76041c1db6467E7b37b977dD7

execution cost     24971 gas (Cost only applies when called by a contract)

hash               0x5364225f1242e5e406688d90d196952a1367868eb9330cf24a57eaa9fdf716cf

input              0x22b...00000

decoded input      {
                          "string _devID": "devID"
                   }

decoded output     {
                          "error": "Failed to decode output: Error: overflow (fault=\"overflow\",
                   operation=\"toNumber\",
                   value=\"39638773911973444535759830463481156742217007468207535463315343515080657469 44\",
                   code=NUMERIC_FAULT, version=bignumber/5.5.0)"
                   }
```

Figure 5.22 Get Device Status Transaction Failure Logs

5.4 Networks Setup

5.4.1 Blockchain and IPFS Network

This section explains how Blockchain and IPFS networks were set up. There are various ways of setting up the Blockchain network and these were explained in Section 2.2. One of the ways explained is based on the third-party services which are services consisting of the nodes that can be utilized to access the Blockchain network. Note, instead of setting up your Blockchain nodes or network manually, one can simply set up the network via third-party services. The proposed solution utilizes a third-party service called Infura.

Instead, of creating the custom node one can register to an IPFS service that offers a node that is already connected to the network which can be accessed via the API. The node gives access to different Ethereum networks such as Rinkeby, Ropsten, Kovan, and even the Ethereum main network. In this study, the infura node is used to connect to the Rinkeby network where our contracts were deployed. For accessing the IPFS network, the custom IPFS node was not used but instead, the infura service was used to access the network. Once both Blockchain and IPFS nodes were set up, the smart contract can be implemented, and made available publicly by deploying them to the public Ethereum Blockchain networks. The RPC calls were made to interact with the deployed contract to the network.

The Blockchain and IPFS nodes fulfill the requirement of REQ5. These nodes ensure that the stored data is distributed in multiple places eliminating a single point of failure thus the high availability of data is achieved

5.4.2 LoRaWAN Network

This section explains the LoRaWAN network setup, which includes setting up the Low-powered devices, LoRa gateway, and LoRaWAN servers.

LoRa Node

The Pycom Expansion Board is a development board that operates as a shield for the LoPy model. The Expansion Board provides additional hardware features for the modules attached to it. The features comprise powering the LoPy via a USB port, additional storage for microSD cards, and enabling serial communication. Figure 5.23 and Figure 5.24 show the Lopy4 and Pycom Expansion Board 3.0 respectively. The LoPy is equipped with several connection methods which include LoRa, Sigfox, Wi-Fi, and Bluetooth. It has an Espressif ESP32 chipset, and Semtech LoRa transceiver SX1276 for transmitting LoRa packets and supports the 433 MHz, 510 MHz, 868 MHz, and 915 MHz frequencies. During the firmware update process, only the LoRa interface was utilized whereas other interfaces were not active. Throughout the experiments, the LoPy devices were battery-powered with the LiPo battery attached to the battery connectors. Table 5.3 summarizes the specifications of the LoPy device.

Table 5.3 LoPy device specifications

Details	Specifications
RAM	4 MB

Storage	8 MB
CPU	Espressif ESP32 chipset
Model	LoPy4
Frequencies	433 MHz, 510 MHz, 868 MHz, and 915 MHz
Number of Devices	2

LoPy was combined with the expansion board 3.0 as shown in Figure 5.25 and set up to operate in the European region on 868 frequencies. The LoRa channels were randomly selected, and servers were set up to operate in similar channels along with the devices. The LoPy was powered by a battery of 3.7 V supplying 1200mAh of current. LoPy4 is programmed by using the constraint version of python suitable for the constrained devices which is micro-python. We intended to measure the power used by the end device during the firmware updates therefore the multi-meters were attached to measure both current and voltage as shown in Figure 5.26. The current is measured in series with the circuit whereas the voltage was measured in parallel with the series. The end devices need to be registered to LoRaWAN servers for them to join or form part of the LoRaWAN network. The LoRaWAN servers utilized are explained in the next section.

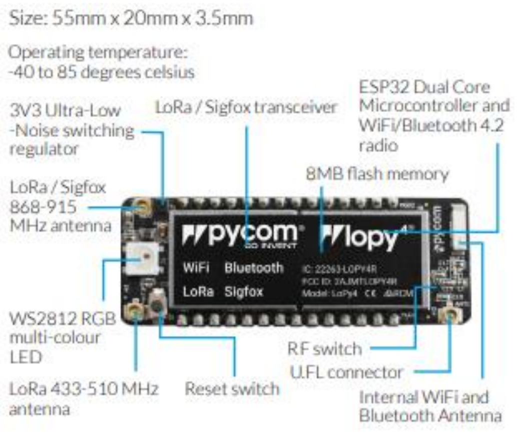

Size: 55mm x 20mm x 3.5mm
Operating temperature: -40 to 85 degrees celsius
3V3 Ultra-Low -Noise switching regulator
LoRa / Sigfox transceiver
ESP32 Dual Core Microcontroller and WiFi/Bluetooth 4.2 radio
8MB flash memory
LoRa / Sigfox 868-915 MHz antenna
WS2812 RGB multi-colour LED
RF switch
U.FL connector
LoRa 433-510 MHz antenna
Reset switch
Internal WiFi and Bluetooth Antenna

Figure 5.23 LoPy LoRa Node.

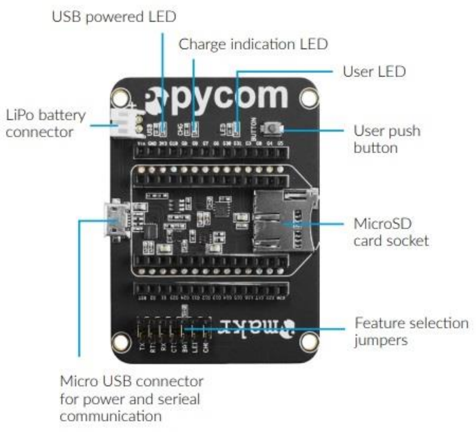


Figure 5.24 The Expansion Board 3.0.

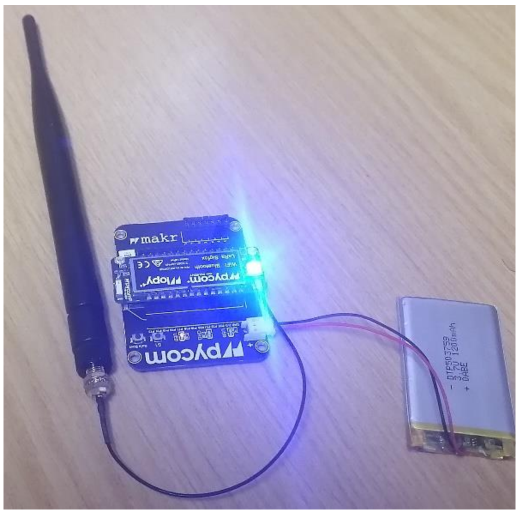


Figure 5.25 LoPy Attached to the Expansion Board 3.0.

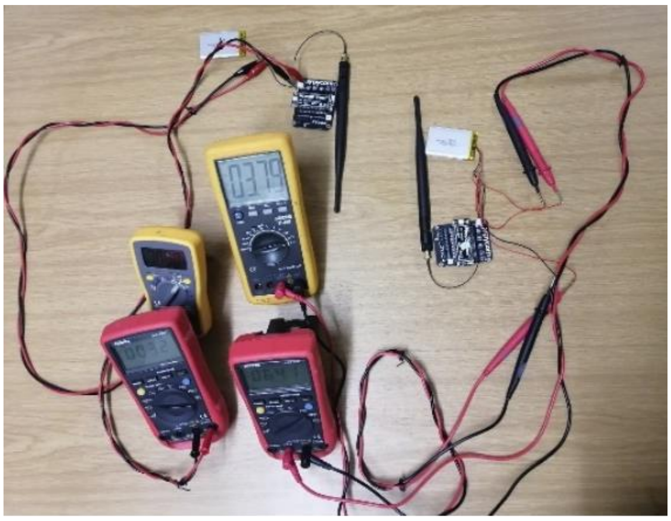


Figure 5.26 Low-Powered Devices with Multi-Meters Attached.

LoRaWAN Gateway and Servers

LoRa devices need a way to receive firmware fragments. It should be noted, it cannot connect directly with the network servers since they do not have LoRa interfaces therefore, the LoRa gateway is utilized to connect LoRa devices with the servers. The Gateway comprises the RAK831 concentrator module, Raspberry Pi B Model, and Antenna that supports 868.1 MHz frequency as shown in Figure 5.29. The created gateway is then utilized to handle LoRa packets sent between the devices and the servers. The gateway is only responsible for sending LoRa packets thus, there must a component that is responsible for filtering packets, removing duplicates of messages, performing encryption, and more.

Table 5.4 Computer Specification

Specifications	Description
Processor family Processor model	4th gen Intel® Core™ i5 i5-4300M
Processor frequency	2.6 GHz
RAM	8 GB
Storage media	SSD

The LoRaWAN servers help to perform such operations. Many available LoRaWAN servers are being developed for his purpose. The Things Network stack community (version 3.13.2) was chosen to handle LoRa packets. The TTN stack was installed in the Lenovo computer shown in Figure 5.30 with Intel® Core™ i5-4300M CPU @ 2.60GHz, 8GB of RAM with Ubuntu 20.04 as an Operating System.

The stack was hosted locally meaning that the private LoRaWAN network was created instead of utilizing the existing public Things Network. Note that after the successful setup both Low-powered devices and the gateway need to be registered first to the servers before being able to exchange any LoRa packets with the servers. The TTN stack provides different integration that helps to process data and trigger events. One of the integrations that the proposed work used is an MQTT integration. The stack exposes an MQTT server to be able to create the MQTT client that subscribes to messages coming from the Low-powered devices and can schedule the downlink messages to the end devices. Many other integrations can be used, such as gRPC, and HTTP, but MQTT is being utilized for this study. MQTT is a lightweight protocol made for IoT devices. Other protocols are more suitable for the exchange of messages between devices that do not have resource constraints e.g., gRPC, and HTTP. When it comes to MQTT and HTTP. MQTT ensures high delivery guarantees of messages, has a low power consumption, provides an open connection between devices, and is mainly used for sending a small message. HTTP on the other hand has high power consumption, opens and closes the connection for every request made, and is mainly useful for sending large messages. In this study, MQTT is used because we always want to keep the connection open between the LoRa end devices/LoRaWAN server and the FUS to constantly send and receive firmware fragments. Moreover, the messages being sent are of t a small size which makes it even more suitable compared to HTTP integration.

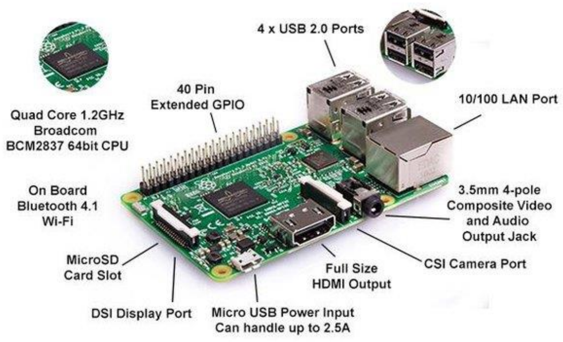

Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
4 x USB 2.0 Ports
40 Pin Extended GPIO
10/100 LAN Port
On Board Bluetooth 4.1 Wi-Fi
3.5mm 4-pole Composite Video and Audio Output Jack
MicroSD Card Slot
CSI Camera Port
Full Size HDMI Output
DSI Display Port
Micro USB Power Input Can handle up to 2.5A

Figure 5.27 Rapberry Pi 3 B Model.

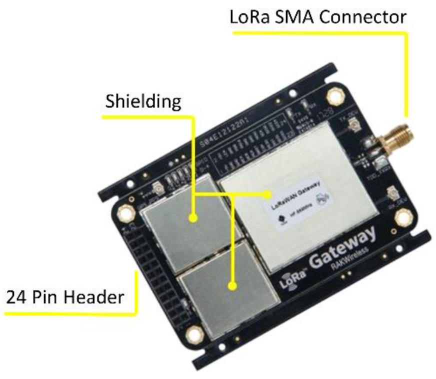


Figure 5.28 RAK832 LoRa Module.

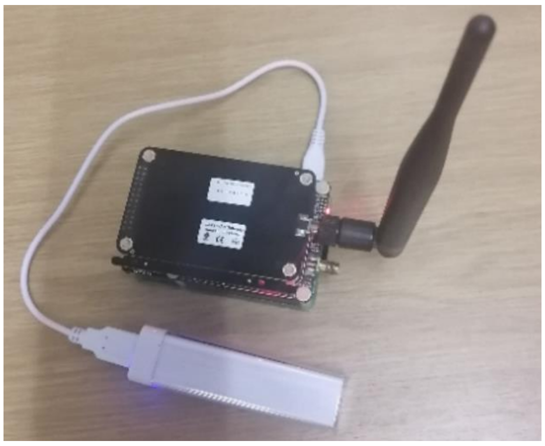


Figure 5.29 RAK831 LoRa Gateway.

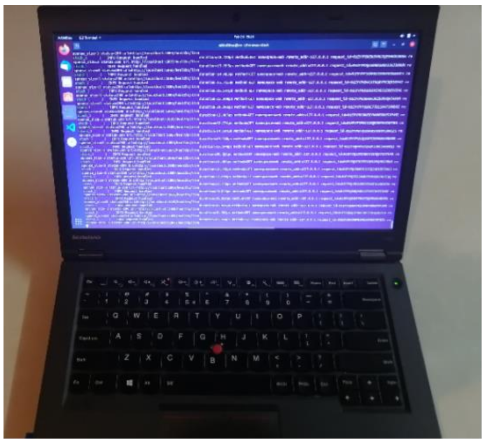


Figure 5.30 Computer Running the TTN Stack.

Table 5.5 Utilized Devices Datasheet

Specifications	URLs
LoPy	https://pycom.io/wp-content/uploads/2017/11/lopy4Specsheet17.pdf (accessed date: 21 November 2021)
Pycom Expansion Board	https://docs.pycom.io/gitbook/assets/expansion3-specsheet-1.pdf (accessed date: 21 November 2021)
Raspberry Pi 3 Model B	https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf (accessed date: 21 November 2021)
RAK831 Module	https://docs.rakwireless.com/Product-Categories/WisLink/RAK831/Datasheet/ (accessed date: 21 November 2021)

The datasheets for LoPy, the Expansion Board, Raspberry Pi, and the RAK831 module can be accessed via the URL provided in Table 5.5.

5.5 Web Application Development

The Web application (Manufacturer UI) was developed to interact with the Blockchain network. The device manufacturers utilize the application to upload both the firmware and the metadata to the IPFS and Blockchain respectively. The web application is connected to the Blockchain and IPFS through the third-party service node called the INFURA. The firmware image must be first uploaded to the IPFS network before uploading the metadata to the Blockchain. This is because the IPFS network will return an IPFS hash that uniquely identifies the firmware being uploaded. The hash also needs to be part of the metadata; therefore, it must be generated first before the metadata upload. The manufacturer UI was developed utilizing these tools:

Web3Js

The web3.js enables the UI to communicate with Blockchain via the provider that does Remote Procedure Call (RPC) to interact with the deployed contract.

IPFS.Js

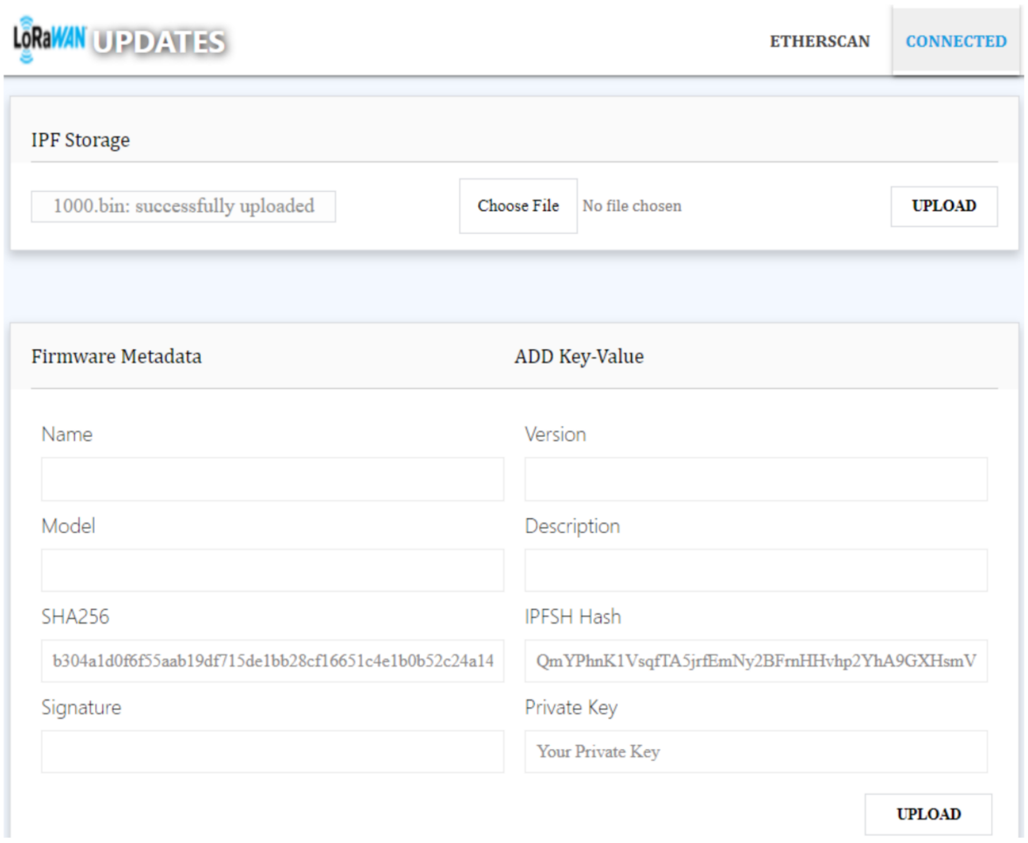
It is a JavaScript library that connects the user interface with the IPFS public network. This library helps to upload the firmware image to decentralized IPFS storage.

MetaMask

When the manufacturer makes a transaction via the web application to the Blockchain. Blockchain identity is required, therefore, Blockchain keys are needed to sign the transactions. MetaMask is used as a wallet to store the private and public keys that sign Blockchain transactions when the metadata is being uploaded.

ReactJS

The front end of the web application was created in ReactJS which is the JavaScript library for building user interfaces.



IPF Storage

1000.bin: successfully uploaded

No file chosen

Firmware Metadata

ADD Key-Value

Name

Version

Model

Description

SHA256: b304a1d0f6f55aab19df715de1bb28cf16651c4e1b0b52c24a14

IPFSH Hash: QmYPhnK1VsqfTA5jrfEmNy2BFmHHvhp2YhA9GXHsmV

Signature

Private Key: Your Private Key

Figure 5.31 Decentralized Web Application for Firmware Upload

```
{
    name: 'LoPy4-firmware',
    version: '1.2.0',
    model: 'LoPy4',
    SHA256: '569948b4baa...',
    IPFS_HASH: 'QmaY7aKo...',
    Signature: 'Ed30Ac8a...',
    ....
    ....
    ....
}
```

Figure 5.32 Structure Example of the Metadata

The developed UI shown in Figure 5.31 takes values from the fields and represents them in a key-value pair. The UI enables the manufacturers to dynamically create the key-value pairs which form part of metadata. The example structure of the firmware metadata is shown in Figure 5.32.

5.6 The FUS Implementation

The FUS is the core component that interacts with Blockchain and is responsible for the entire update process. This section views how the FUS was implemented together with the libraries used to implement it and how it addresses some of the requirements stated in Section 4.3. Just like the web application presented in Section 5.5., the FUS also connects to both IPFS and Blockchain networks and additionally to the LoRaWAN network. Figure 5.33 shows how FUS interacts with these networks.

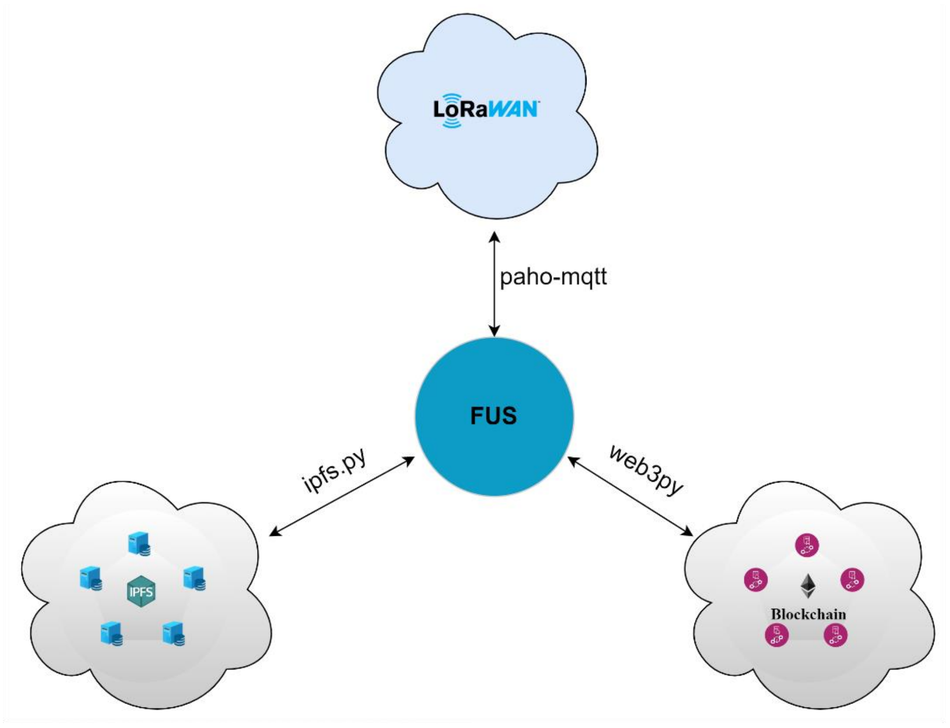


Figure 5.33 FUS Utilized Libraries.

The FUS was implemented in python and utilizes the libraries shown in Figure 5.33. The libraries help to connect the FUS to the IPFS, LoRaWAN, and Blockchain network.

IPFS.Py for IPFS

The FUS utilized the ipfs python module which connects the FUS via HTTPS connection during the firmware update to retrieve firmware images. The FUS does not connect to the locally IPFS node to get access to the network, however, it uses the third-party service node to get access to the public network. From this python module, we only utilize the get method which retrieves the firmware image from the IPFS decentralized file storage.

PAHO-MQTT

The FUS implements the MQTT client which directly connects with the LoRaWAN application via the exposed MQTT broker. It uses the paho-MQTT python library to create the MQTT client that connects to the application server.

Blockchain network – The FUS interacts with the Blockchain by utilizing the web3 library that is implemented in python. It also gets access to the Blockchain via the infura node and uses the service API keys to make RPC calls to the Blockchain. Web3 library enables the FUS to connect via the HTTPS and the WebSocket. When the FUS is connected via WebSocket, it becomes easy to listen to Blockchain events in real-time.

Chapter 6: Results and Discussion

This chapter provides an evaluation of the developed and implemented system. It starts by introducing a STRIDE thread model and analyses all possible weaknesses of the proposed solution and associates them with the mitigations that were adopted by the solution. The overall setup of our experiment together with the parameters used is also explained then after the costs of cryptographic techniques on low-powered devices are examined. Afterward, the analysis of the costs involved when updating low-powered devices in the Long-Range Wide Area (LoRaWA) network is provided, with the Blockchain operation costs involved during the update process. Finally, the chapter provides comparisons between the state-of-the-art solutions and our proposed solution.

6.1 Security Analysis: Threat Assessment

In this study, we performed a security analysis of our system. We performed threat modeling on selected components and data flows which are illustrated in Figure 6.1. These components include the manufacturer's user interface, Firmware Update Service (FUS), IPFS (INFURA service), Blockchain, LoRa Servers, IoT gateway, and IoT device. It is important to note that the threat analysis for some components including IPFS networks and Blockchain networks was kept out of scope but their interaction with important system components was examined. The reason for this is that these components belong to providers, and it is believed that the providers take appropriate countermeasures for possible security issues as they are accounted for. This research considered only threats that are associated with major components and data flows.

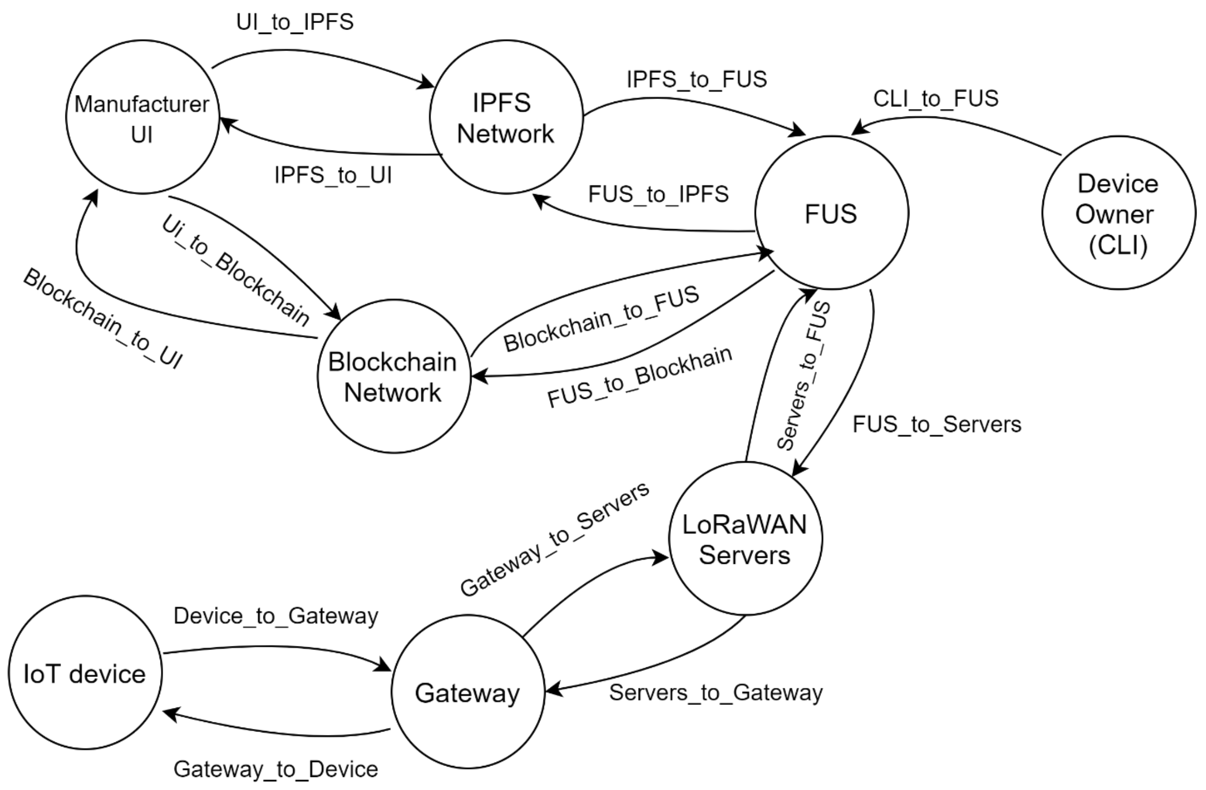


Figure 6.1 Data Flow Diagram of Components of Proposed Architecture.

6.1.1 Threat Models

This research utilized the Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, and Elevation of Privilege (STRIDE) threat Model which is an industry-standard for evaluating systems regarding security. STRIDE is considered a lightweight approach compared to the Process for Attack Simulation and Threat Analysis (PASTA), Operationally Critical Threat Asset and Vulnerability Evaluation (OCTAVE), and Common Vulnerability Scoring System (CVSS) which are other modeling methodologies.

The choice of STRIDE is motivated by several reasons:

- STRIDE is widely accepted in industry and academia.
- It analyses security properties such as authentication, integrity, confidentiality, authorization, availability, and non-repudiation which are security properties that the proposed architecture desires to achieve as well.

STRIDE provides analyses of cyber threats against each system component based on its technical knowledge and provides a clear understanding of the impact of a component vulnerability on the entire system. Figure 6.2 explains the STRIDE model terms.

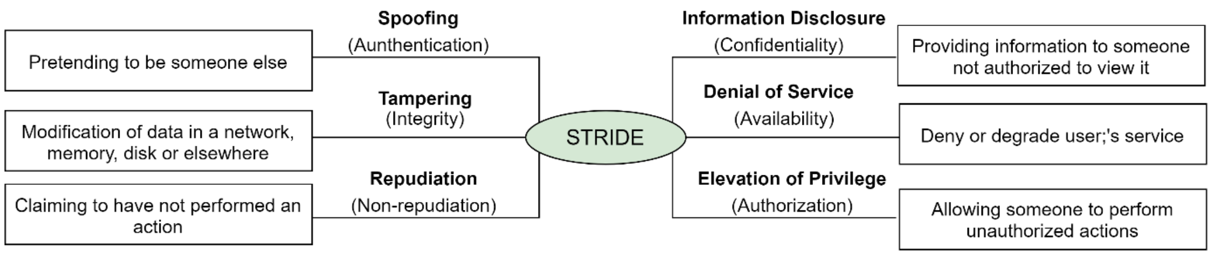

Pretending to be someone else
Spoofing
(Aunthentication)
Modification of data in a network, memory, disk or elsewhere
Tampering
(Integrity)
Claiming to have not performed an action
Repudiation
(Non-repudiation)
STRIDE
Information Disclosure
(Confidentiality)
Providing information to someone not authorized to view it
Denial of Service
(Availability)
Deny or degrade user;'s service
Elevation of Privilege
(Authorization)
Allowing someone to perform unauthorized actions

Figure 6.2 STRIDE Threat Modelling (Azam *et al.*, 2022)

6.1.2 Identified threats and Defense Mechanism Discussion

A data flow diagram for all components of the proposed firmware architecture was presented in Figure 6.1. This section identifies the possible threats and provides the defense mechanisms used to eliminate the threats.

I to IPFS and IPFS to UI

The attacker can get the manufacturer's UI and try to upload the firmware image to the IPFS network if the UI is open and perform a MITM attack to tamper with the firmware image while it is in transit or before it reaches the IPFS network. On the other hand, the IPFS returns the IPFS hash to the UI which later forms part of the metadata thus, when the hash returns the attacker can perform a MITM attack to view the hash (Information Disclosure) and possibly tamper it.

UI to Blockchain and Blockchain to UI

The attacker could upload badly signed firmware metadata as well as sniff and obtain legitimate firmware metadata. Metadata transmission could be susceptible to tampering, information disclosure, spoofing, and elevation of privileges. Spoofing the manufacturer's identities could be possible if an attacker somehow knows a manufacturer's identity keys and can act as a manufacturer. When the metadata reaches the Blockchain network specifically the smart contract, it can be susceptible to an elevation of privilege, enabling an attacker to store unauthorized firmware metadata on the smart contract.

FUS component, IPFS to FUS, and FUS to IPFS

One of the actions that the FUS performs during these processes is to download the firmware image from IPFS. The FUS controlled by an attacker could download a firmware image and the attacker could perform a MITM attack and force a FUS to download the image. This violates integrity and confidentiality and the tampering and information disclosure get affected.

CLI (Device Owner) to FUS

The CLI request tool and FUS controlled by an attacker could initiate a firmware update procedure causing the Denial of Service of the FUS and if knows the attacker knows authentication keys can spoof the FUS.

Blockchain to FUS and FUS to FUS

There is an open web socket between Blockchain and the FUS. If an attacker eavesdrops on the connection when it is not secured, then tampering and information disclosure could take place. In addition, if the user is in control of the FUS during that particular moment of firmware metadata transmission, an attacker could authenticate (spoof) the metadata.

FUS to Servers to Gateway and IoT Device

An attacker in control of the FUS could perform a Denial of service and replay downlink messages to the LoRaWAN servers. The FUS can be spoofed and a MITM attack could occur where the session keys eavesdrop which results in tampering and information disclosure. This could occur during the FUS to Server, Servers to Gateway, or Gateway to IoT device interaction. Subsequently, the Elevation of Privilege could take place if illegitimate downlinks and replayed session keys are allowed by the architecture. It can be noted that from IoT devices to gateway, server, and FUS the Elevation of Privilege, Spoofing, tampering, and information disclosure is also possible. The attacker in control of the device can do several things, assuming he has control of the cryptographic keys. It is possible to tamper with information like session keys, firmware images, and signatures. Moreover, an attacker could send back the confirmation message to the FUS spoofing it as if the device was successfully updated.

Thread Category	Defense
Spoofing	Elliptic Curve Digitial Signature Algorithm (ECDSA), HMAC-SHA256, RSA keys, Use of nonce values for the elimination of replay attack
Tampering	HMAC-SHA256, SHA256
Repudiation	Logging
Information Disclosure	Encryption: used Advance Encryption Standard (AES), RSA keys
Denial of Service	Data Rate restriction by the network, Multiple nodes storing data
Elevation of Privilege	Access Control Authorization e.g. through a smart contract, cryptography

Considering the STRIDE analysis, it is believed that the usage of encryption, authentication, and integrity mechanisms in our proposed architecture such as AES, HMAC algorithm, SHA256 algorithm, ECDSA algorithm, RSA certificates, logging of the firmware update process, and the use of nonce values for spoofing elimination can defeat all mentioned attacks. This is true if legitimate entities such as the manufacturer and the device owner retain their cryptographic keys and are not exposed to be obtained by an attacker. We can then conclude that ours is a system safe and that it is protected against all possible attacks.

6.2 Evaluation Metrics

The previous chapter presented the setup of each network namely the LoRaWAN and decentralized network setup (Blockchain and IPFS). Figure 6.3 summarizes the overall experiment setup of these networks. This section aims to present the parameters used in the evaluations and the evaluation metrics and how each was measured.

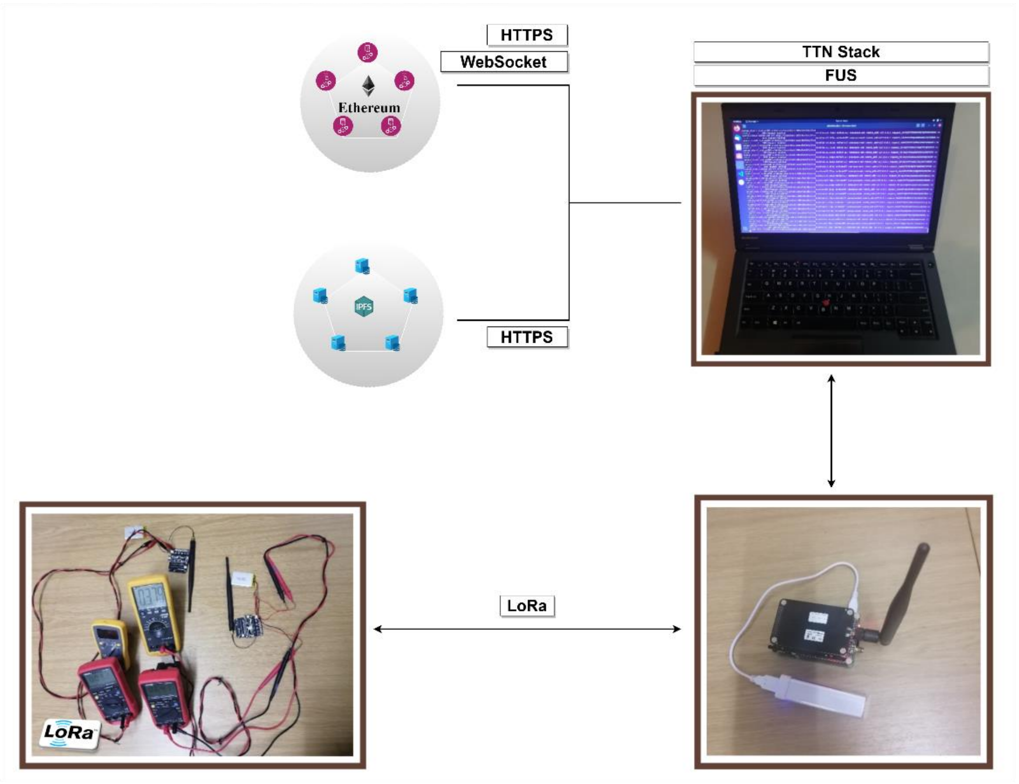


Figure 6.3 Experiment Architecture.

Table 6.1 Experiment Parameters

Parameters	Description
Receive Window (Rx1)	This receive window is used for both downlink and uplink messages

Receive Window (Rx2)	The Rx2 window is used for downlink messages utilizing 869.525 frequency
Region	EU (8 channels with a duty cycle of 1% and one with 10%)
Gateway(s)	1
LoRa devices/Low-powered	2
Devices Modes	Class A and Class C modes. Devices were also set to operate in multicast
Rx1, Rx2 delay	5s, 1s delay respectively
Bandwidth	LoRa.BW_125KHZ
SFs	7-12

Table 6.1 shows the experimental parameters used in the evaluation:

- Class Mode: The architecture is evaluated and tested against Class A and Class C modes. For Class A mode, the device needs to send the uplink message before it can receive the firmware fragment. In addition, the architecture is evaluated against multicast sessions where the end device operates in Class C mode which is always listening to receive firmware fragments.
- RX1, RX2, and RX delay: The firmware fragments are received in both RX1 and RX2 windows. The RX2 window/channel comprises 10% duty cycle restrictions, whereas the other channels comprise 1% duty cycle restrictions (per sub-band).
- Gateway and end devices: Only one gateway and two end devices are utilized. Both the end device and the gateway respect the duty cycle restrictions.
- Region and Bandwidth: The low-powered devices are equipped to operate in the European region; this modulation operates in the radio band 863–870 MHz, with a bandwidth of 125 kHz.
- SFs: LoRa SF ranges between 7 and 12. The SF impacts the communication performance of LoRa. The architecture will be tested on different SF to see how the low-powered devices perform during the firmware update on these SFs.

The evaluations were divided into three phases: The initial phase started by looking at the security aspect of low-powered devices. This includes examining the memory consumption utilized by cryptographic algorithms during the update process. The second phase looks at the LoRaWAN evaluations which include examining the different modes of classes of low-powered devices, SFs, airtime, update time, and the effects of firmware fragments. The third phase looked at the Blockchain-based costs of the operations of the proposed contract. The evaluation results are presented in terms of the following metrics:

- **Memory consumption** indicates the amount of RAM and Flash memory consumed by low-powered devices. This metric was measured by running a python script that calculates the memory consumption for a specific firmware image update.
- **Power consumption** indicates the power consumption of the devices during the firmware update process. This metric was measured using the two multi-meters of each device. One multi-meter was used to measure the current and the other the voltage at the epoch during the firmware update.
- **Update time** indicates the time it takes for a device to successfully receive all firmware fragments and includes the time to verify the firmware. This metric was measured by running a python script that utilizes the timer to calculate the time it takes for a firmware update to complete,
- **Gas used** indicates the amount of gas used from the provided gas limit. The metric was measured or obtained in two ways. The gas used for a transaction is part of the transaction data and this data can be obtained on the blockchain network by providing the transaction hash. The second way we measured this metric is by running a JavaScript code that obtains the Blockchain transaction data that then extracts the gas used from the data.
- **Gas fee** indicates the amount of Ether charged during the transaction of an operation. The gas fee is obtained the same way as the gas used metric where it is obtained by visiting Blockchain explorer or running JavaScript instructions that are used to obtain it.

6.3 Cryptographic Costs on Low-Powered Devices

To determine whether the proposed architecture suits the constrained devices, the cost of cryptographic algorithms used to secure the firmware during the update process needs to be examined. RFC has defined the categories of constrained devices (Bormann, Ersue, and Keranen, 2014) shown in Table 2.1 to help us determine whether the proposed architecture suits the constrained devices in the network or not. Thus, the device's memory consumption cost which is RAM and flash memory needs to be examined. Figure 6.4, Figure 6.5, and Figure 6.6 show the CMAC, HMAC-SHA256 RAM consumption, and flash memory consumption, respectively. The low-powered device takes 96.7 kB of flash memory when being updated with 5 kB of the firmware. This memory consumption comprises the firmware size and the firmware code during the verification process.

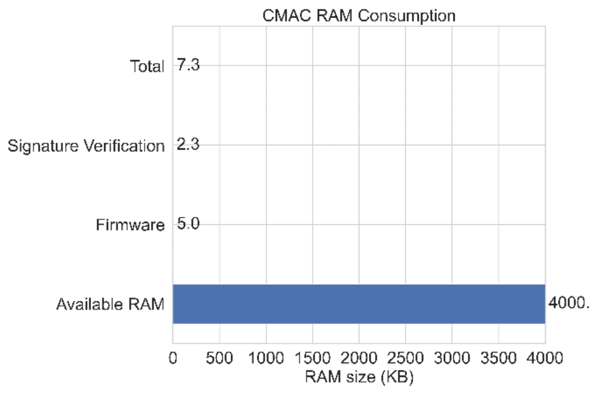


Figure 6.4 CMAC RAM Consumption.

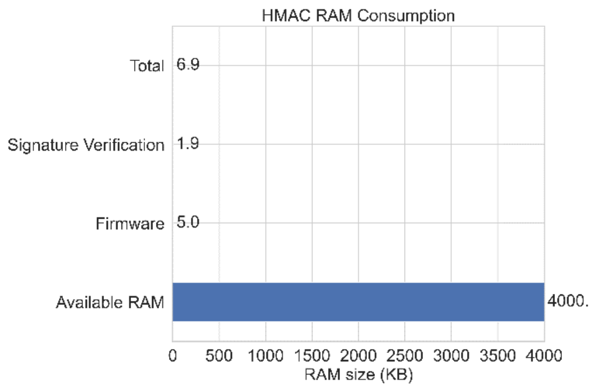


Figure 6.5 HMAC RAM Consumption.

The RAM consumption varies due to the firmware size being updated and the algorithm being utilized to verify the firmware. The results show that the HMAC-SHA256 algorithm verifies faster and consumes less memory compared to the CMAC algorithm. This is due to the implementations of the HMAC-SHA256 algorithm. The HMAC-SHA256 is based on the hash function and hash functions are considered to be faster than block ciphers. Another reason for it being faster than the CMAC algorithm could be the mode of operation used. The mode of operation used by CMAC is MODE_ECB which is considered slower.

The HMAC-SHA256 consumes 6.9 kB of RAM when 5 kB is updated whereas CMAC consumes 7.3 kB of RAM. The RAM and flash consumption imply that MAC algorithms are adequate in providing security on low-powered devices and are suitable for constrained low-end devices that belong to Class 0, Class 1, and Class 2. This suitability is from the fact that the RAM and flash consumption does not exceed the memory of the constrained device classes.

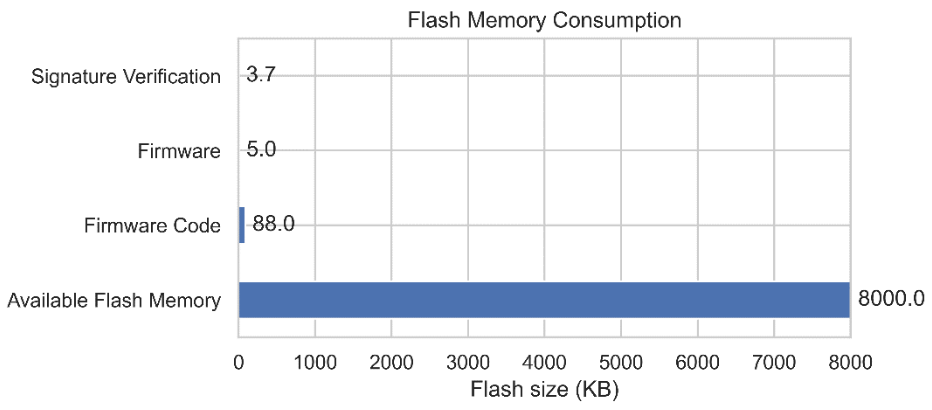


Figure 6.6 Flash Memory Consumption on 5 kB of the Firmware.

We now look at the energy consumption of cryptographic techniques. Measuring the energy consumption is so important because these devices operate on a battery, and it is required to know how much energy is being consumed during the firmware updates. The energy consumption is obtained by a formula:

$$E = \mathrm{I} \times \mathrm{V} \times \mathrm{T} \tag{1}$$

where I represent the current, V represents the voltage and T represents the time in seconds. The current and voltage are measured using the multi-meters demonstrated in Figure 5.26. Figure 6.7 and Figure 6.8 demonstrate the energy consumption results obtained for the cryptographic algorithms used in the update process and also show their update time respectively.

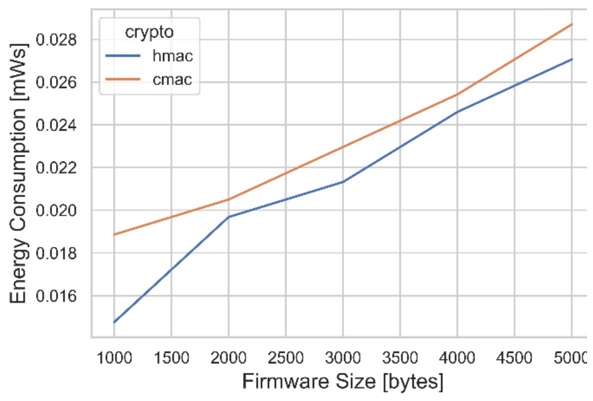


Figure 6.7 Energy Consumption of Cryptographic Algorithms.

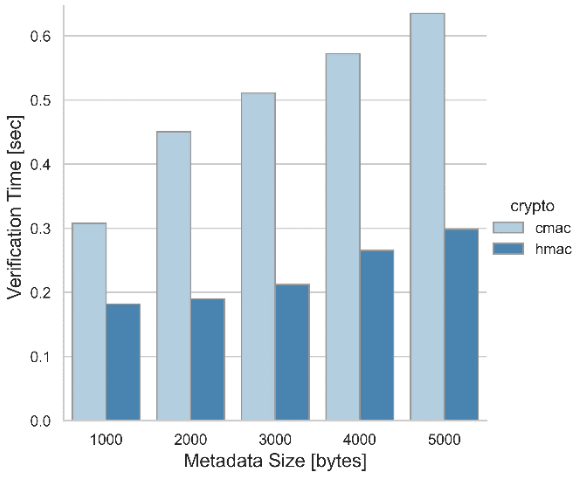


Figure 6.8 Cryptography Verification Time.

The results show that the HMAC consumes less energy and takes less time compared to the CMAC algorithm. This can be because the HMAC id is faster than the CMAC in this case due to the encryption differences utilized by each. As of this result, it can be noted that if the firmware size increases directly proportional to the verification time, memory consumption, and energy consumption. It is then recommended to apply a delta update in these devices instead of sending the larger firmware image, which may consume less memory and less energy.

6.4 Evaluating LoRaWAN Costs

Figure 6.9, Figure 6.10, and Figure 6.11 demonstrate the relationship between firmware size, fragment size, airtime, and SF. When the fragments are sent at a lower SF, it decreases the airtime of the fragment whereas when sent at a higher SF, the airtime increases. It is also observed that increasing the SF by one step for example, from SF11 to SF12 doubles the airtime as it is shown in Figure 6.11. Note: at higher the SFs, the higher the airtime even for the same fragment size. For example, for the same fragment sizes at SF11 and SF12, the airtime tends to differ and SF12 has the higher airtime. Figure 6.9 shows that the produced number of fragments depends on the SF. The higher SF produces a greater number of fragments e.g., if sending 5 kB of firmware with SF12, the number of fragments is 108 whereas with SF7 the fragments produced are 24. This is due to the regional restrictions of LoRaWAN.

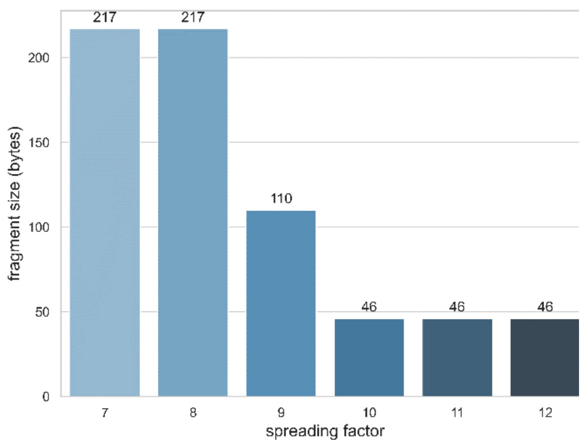


Figure 6.9 SF with Chosen Fragment Size.

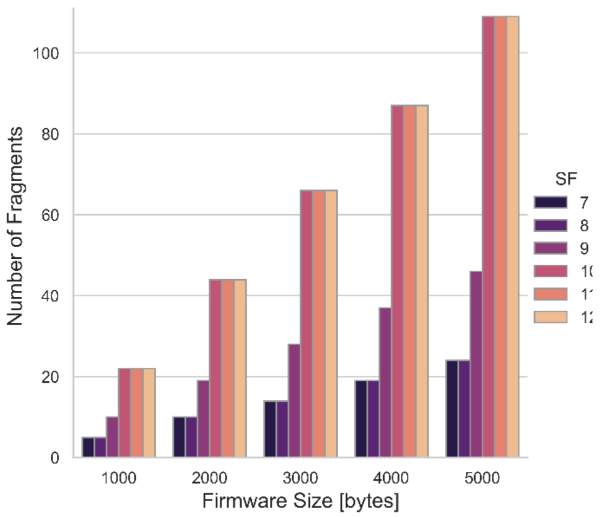


Figure 6.10 SF with the Number of Fragments.

Since the low-powered devices operate in the European region, the maximum number of payloads must not be exceeded. For instance, when the end device is operating with SF12, SF11, and SF10, the fragment size must be less than 51 bytes. For SF9 the fragment size must not exceed 115 bytes and for SF8, and SF7, the fragment size must be less than 222 bytes for efficient transmission. Figure 6.9 depicts the chosen number of fragment sizes for each spreading factor, and the chosen fragment sizes adhere to the European Region (EU).

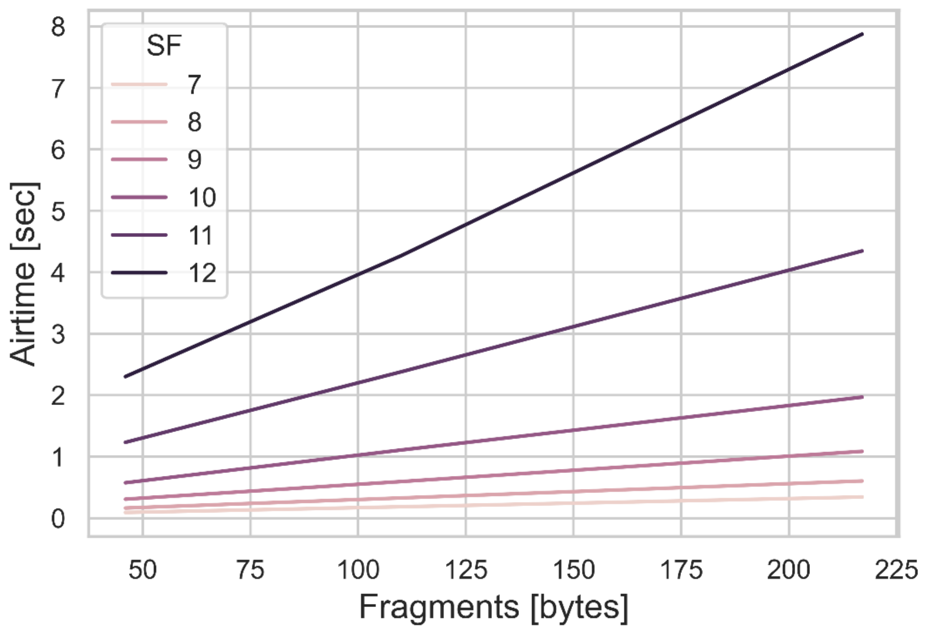


Figure 6.11 Airtime and Fragment Size.

Now looking at the time it takes to update LoRa end devices. Figure 6.12, Figure 6.13, and Figure 6.14 show the update time, which starts from where the firmware update is initiated until the firmware image is verified.

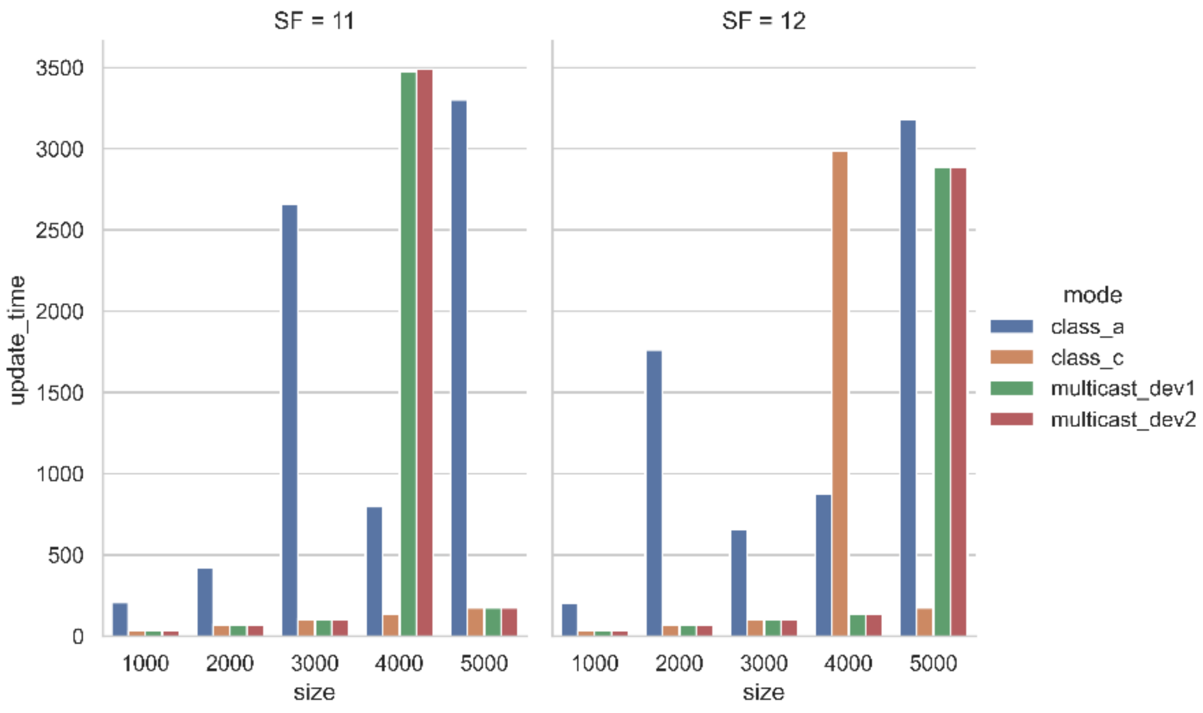


Figure 6.12 Update Time for SF12 and SF11.

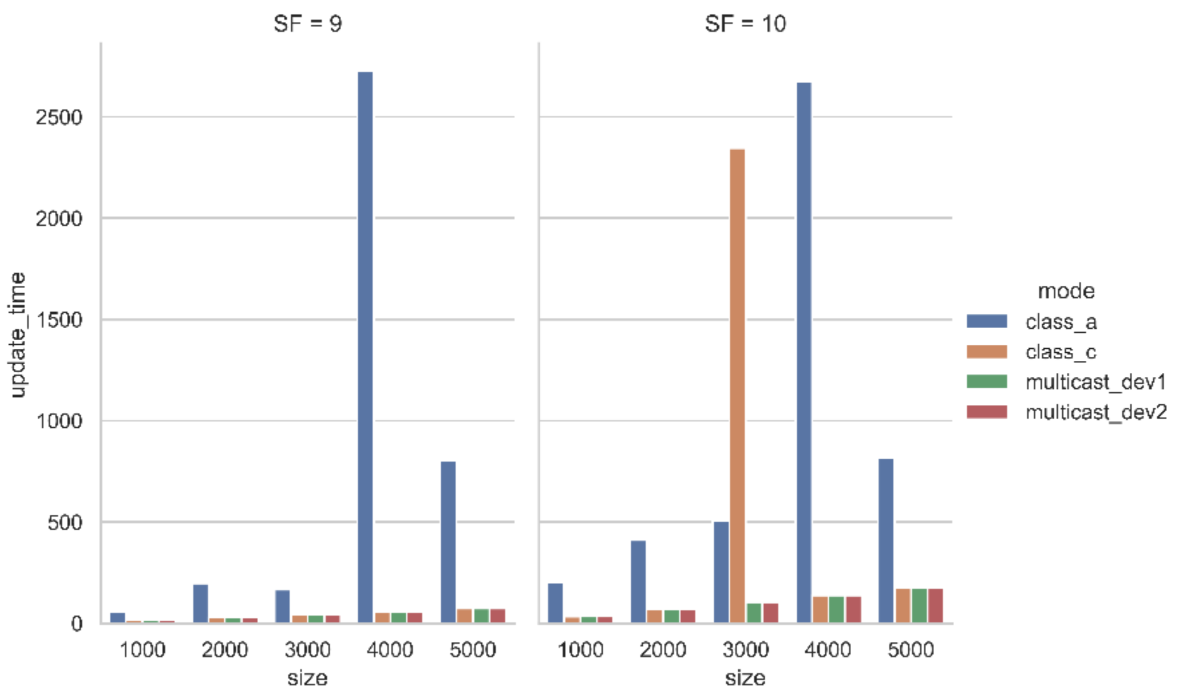


Figure 6.13 Update Time for SF10 and SF9.

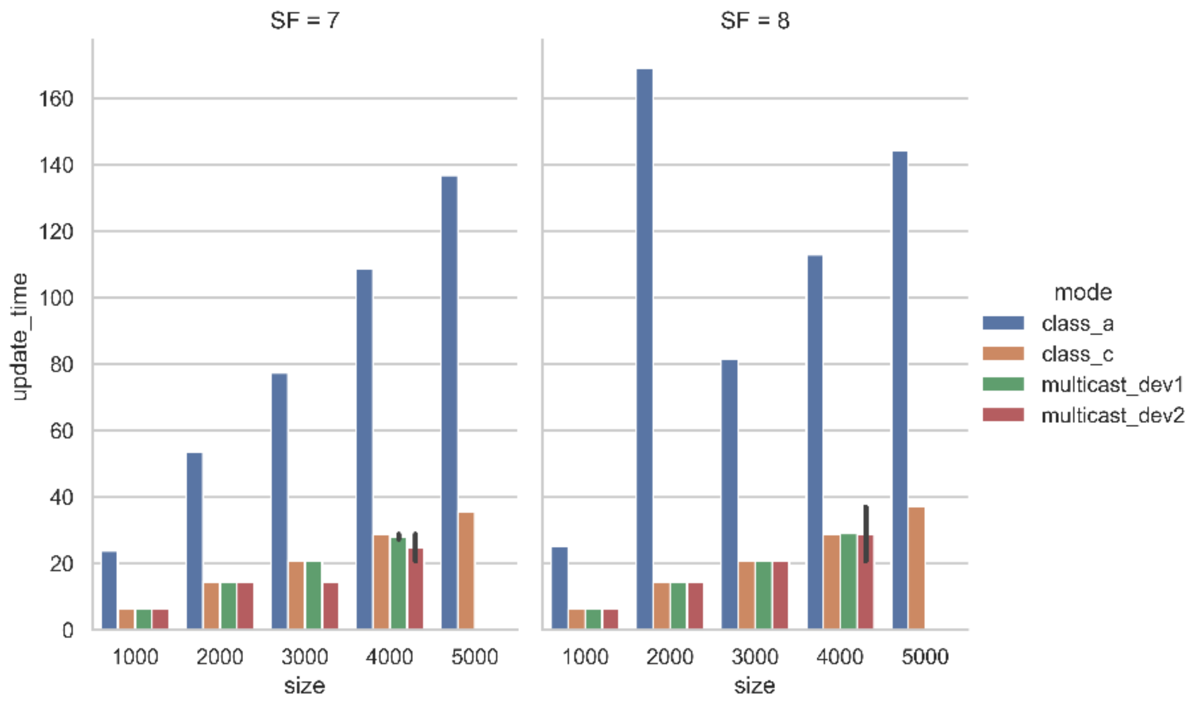


Figure 6.14 Update Time for SF8 and SF7.

The two modes of LoRa end devices were examined which are Class-A and Class-C. The update time, firmware size, and the SF show a directly proportional relationship. For example, updating operating in the Class-A mode with 4 kB of firmware at SF 11 takes more time, i.e., 796.10 seconds (13 minutes), compared to updating the firmware at SF7 which takes only 108.65 seconds (1.8 minutes).

There are several reasons which could cause the increase in update time. This could be the firmware size and fragment size. The firmware size greatly impacts update time because it needs to be fragmented. Bigger firmware size means many fragments are required to be produced and therefore, more fragments are required to be sent to the end device. It could also be affected by SF and airtime, increasing the SF the update time increases. This is mainly due to the large number of fragments that are produced at higher SFs. And this will result in long airtime. One of the LoRaWAN restrictions is the duty cycle, which impacts the update time.

LoRaWAN limits the maximum application payload that needs to be sent over the channel. This increases the update time because each SF LoRaWAN restricts the payload size, resulting in higher SFs sending more fragments to lower SFs. When comparing the update time of the different modes which include Class-A, Class-C, and multicast, it was observed that when the devices are operating in the Class-A mode, it always takes more time to update the devices compared to other modes. In Class-A mode, the end device needs to send some data before it receives any firmware fragment. Moreover, the network servers set the recommended RX1 delay to 5s for Class-A. This means that the next firmware fragment will be received after 5s. The multicast mode is similar to the Class-C; the difference is sending the single firmware to the set of devices. Figure 6.12, Figure 6.13, and Figure 6.14 show that the update time for both Class-C and multicast is quite the same in some cases. For example, when updating the end device at SF8, the update time is quite the same with all firmware sizes for both Class-A and Class-C, but this does not hold in some cases. The update time is not predictable.

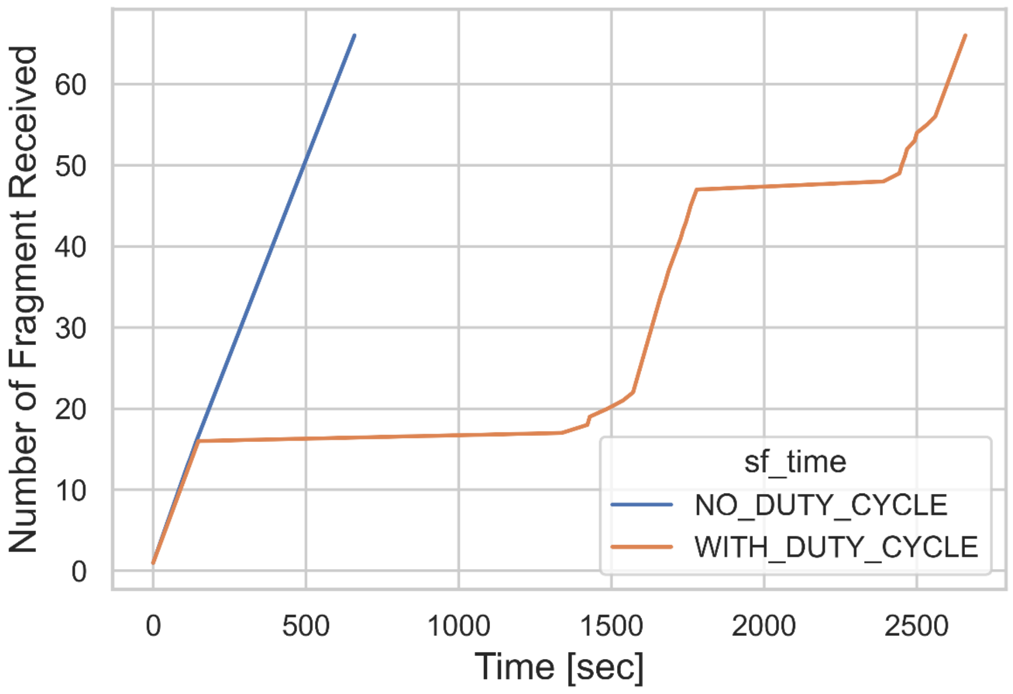


Figure 6.15 Delay in Update Time Due to Duty Cycle Restrictions

This is observed in Figure 6.15 where the device was updated with 3 kB of firmware operating in Class-A mode. It was expected that the SF11 update time should be less compared to SF12, however, that was not the case. This is because there was a time when the end device was inactive, not receiving the fragment for some time. This is due to the LoRaWAN restriction which affected the update time. If the time to deliver the firmware image matters a lot, it is preferable to use the lowest SF, i.e., SF7. During the firmware update, it was observed that more firmware fragments were lost when the LoPy was operating on the SF7. The higher SF has the benefit of extended airtime. It gives better sensitivity or better coverage for the LoRa end device that are further away to receive the firmware fragments however this causes some delays for the end device to be updated. The total number of firmware fragments exchanged is not fixed; it depends on several things which are the class modes the device is operating in, the data rate or SF utilized by the low-powered device, and the number of retransmitted firmware fragments during the update process. Table 6.2 shows the exchanged number of messages when the low-powered device was updated with 1 kB of the firmware at SF12.

Table 6.2 Exchanged Number of Messages at SF12 using 1 kB Firmware.

Description	Number of Messages
Class A uplinks	33
Class A downlinks	25
Class C uplinks	5
Class C downlinks	25

When the device utilized Class A mode, many firmware fragments were exchanged compared to the number of fragments exchanged when the low-powered device utilized Class C mode. this was because, in Class A mode, the low-powered device was always sending an uplink message to receive the firmware fragment, whereas Class C mode required no uplink messages to be transmitted by the low-powered device to receive the firmware fragment. However, for Class C downlink scheduling to start, the activation uplink message needs to be sent by the device to the LoRaWAN network servers, specifically after the OTAA join–accept. The total number of uplinks comprises the activation messages, uplink messages for requesting the retransmission of lost packets, the acknowledge (ACK) messages for both the session keys and metadata, and finally the update status message. The exchanged activation messages are only applicable if the device is operating in Class C mode. The low-powered device sends two activation messages to increase the chances for it to be activated to receive downlink messages. The retransmission query messages are sent in case there were lost firmware fragments. From Table 6.2 there was no query for the retransmission of firmware fragments, the low-powered device successfully received all the firmware fragments. The update status message exchange indicates a successful update status after the verification of the firmware image by the end device.

Note that, for Class A mode, the number of uplink messages produced by the device may increase when the FUS performs other tasks while preparing for a response for the device. For instance, when the FUS receives a confirmation message from the device that session keys were successfully received, the device will keep on sending uplinks while the FUS is busy downloading, verifying firmware and metadata, etc. The number of downlink messages for 1 kB firmware transmission is made up of 22 firmware fragments, the session key exchange message shown in Table 4.4, the metadata exchange message, and the last downlink message designating that all firmware fragments have been sent. The 1 kB firmware was fragmented into 22 fragments because the device was operating with SF12, and Figure 6.9 shows that each fragment size is 46 bytes when the device operates at SF12. Therefore, to read the 1 kB of firmware, it has to be read 22 times.

It is also significant to examine the energy consumption of the low-powered device and the gateway. This depicts energy consumed by the low-powered device when operating in Class A and Class C modes (multicast session uses Class C mode hence, the energy consumption of Class C was only examined). The effect of SFs from 7 to 12 against energy consumption was examined. Figure 6.16 shows that SF12 had the higher energy consumption and SF7 had the lower energy consumption. Energy consumption continues to increase from SF7 for every SF up to SF12. Airtime is one of the factors that cause the increase in energy consumption. Figure 6.11 demonstrated that with the increase in SF, the transmitted fragment requires more airtime to reach the low-powered device hence, there is more energy consumption on the low-powered devices because higher SFs use more chirps for longer transmission.

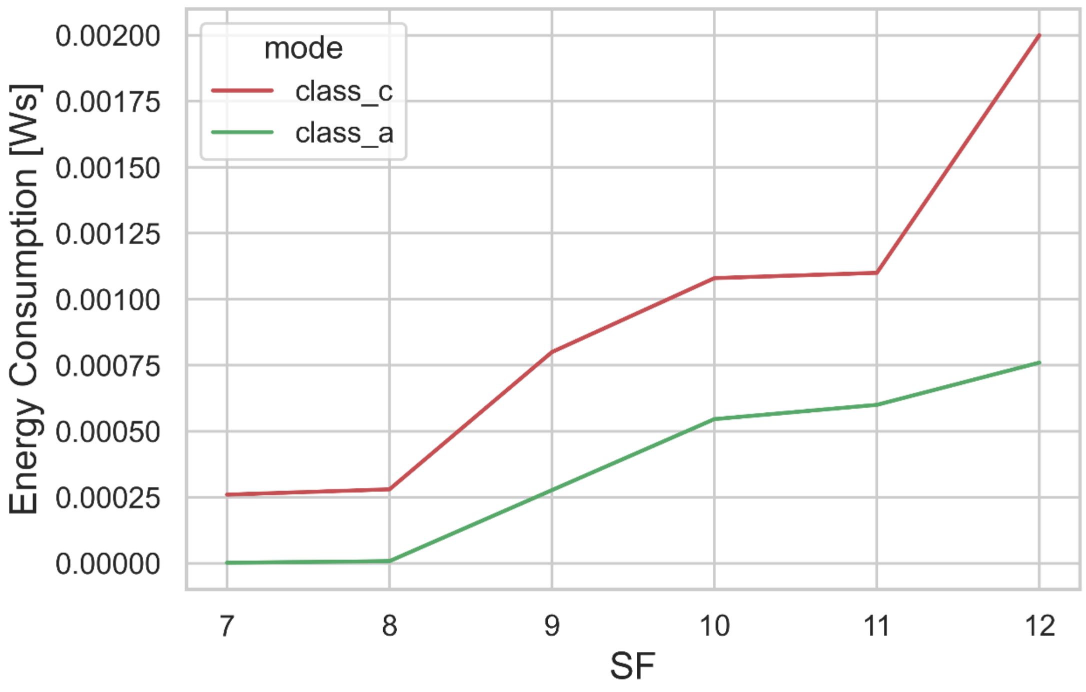


Figure 6.16 Low-Powered Devices Energy Consumption

The energy consumption of the gateway was also examined when the low-powered devices were operating in Class A and Class C modes. Figure 6.16 depicts that Class C mode has higher energy consumption compared to Class A mode. Figure 6.17 shows that the gateway's energy consumption is higher compared to the device's energy consumption. The advantage of a high SF is the extended transmission which gives the receiver more opportunities to sample the signal power, resulting in better coverage; but it consumes high energy compared to other lower SFs.

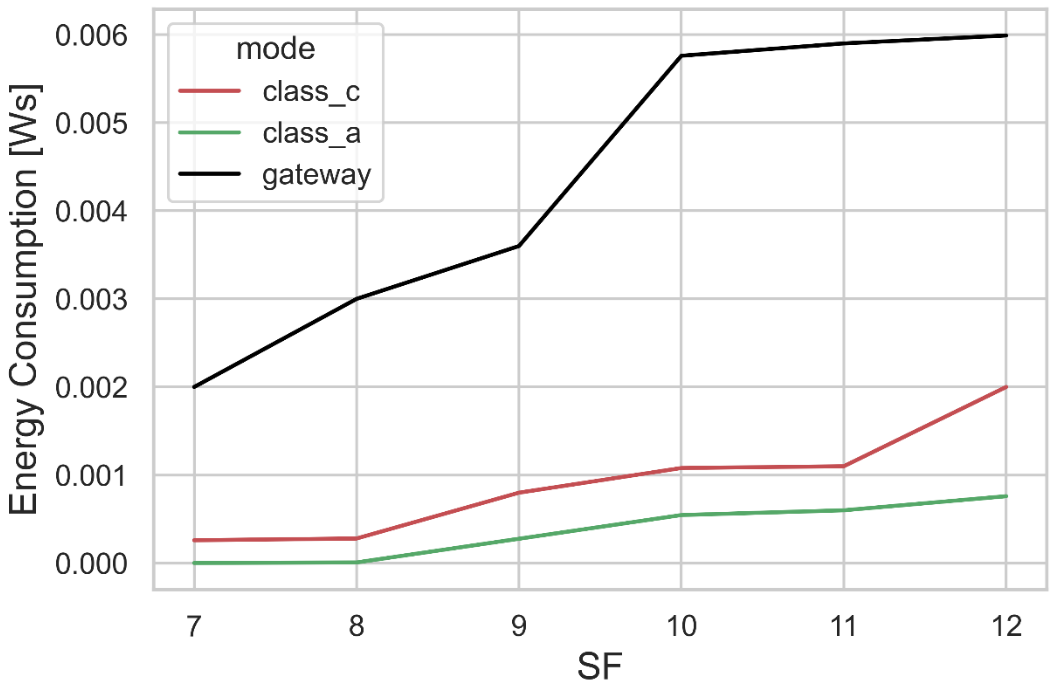


Figure 6.17 LoRa Gateway Energy Consumption

6.5 Blockchain Evaluation Costs

This section examines the Blockchain operation costs involved during the update process. Figure 6.18 and Table 6.3 depict Blockchain operations costs in the Blockchain network which are represented in terms of fee and gas. Each figure depicts that the firmware metadata costs vary from time to time depending on its size. The increase in metadata data size increases the gas cost execution on the Blockchain network. This is illustrated in Table 6.3 where firmware size of 1 kB to 5 kB was used.

Figure 6.18 shows that the reason for an increase in the cost is that the transaction was made when the new firmware metadata was added to the network. In other words, a certain amount of gas or fee is provided by the manufacturer to the network for the successful execution of the transaction. The manufacturer specifies the gas limit which must always be higher than the gas to be used.

Table 6.3 Gas Cost Execution on blockchain Operations

Parameter	Description
addNewFirmware(1 kB)	378,328
addNewFirmware (2 kB)	689,116

addNewFirmware (3 kB)	1089466
addNewFirmware (4 kB)	1176024
addNewFirmware (5 kB)	1282079
registerDevice(devInfo)	49,418
updateDevInfo(devInfo)	28,852
retrieveMetadata()	0
retrieveDevsInfo()	0

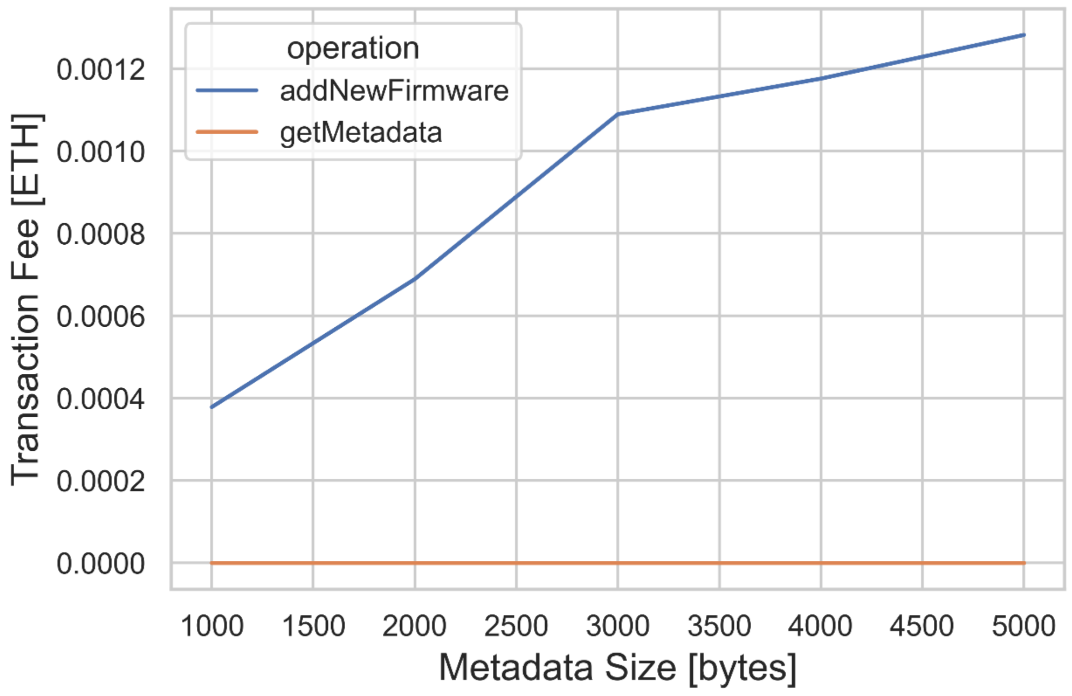


Figure 6.18 Fee Cost on Adding New Metadata and Getting Metadata.

Figure 6.19 demonstrates the relationship between the gas limit and the gas used by the manufacturer when the transaction was executed for storing the metadata.

It is observed that the gas limit must always be higher than the gas used and the manufacturer must ensure that it is enough to cover a transaction otherwise it will not execute successfully. This steadily increasing relationship between fee/gas consumption and metadata is caused by adding more data to the Blockchain, and the fee required to execute the transaction is directly proportional to the amount of data being added. Hence, operations like registering the low-powered device, publishing new metadata, deleting the device, and updating the low-powered device information require gas/fee. Whereas operations like retrieving metadata, and device information require no gas because no transactions are involved. More importantly, the gas/fee consumption may vary from time to time depending on the price at that particular moment on the network.

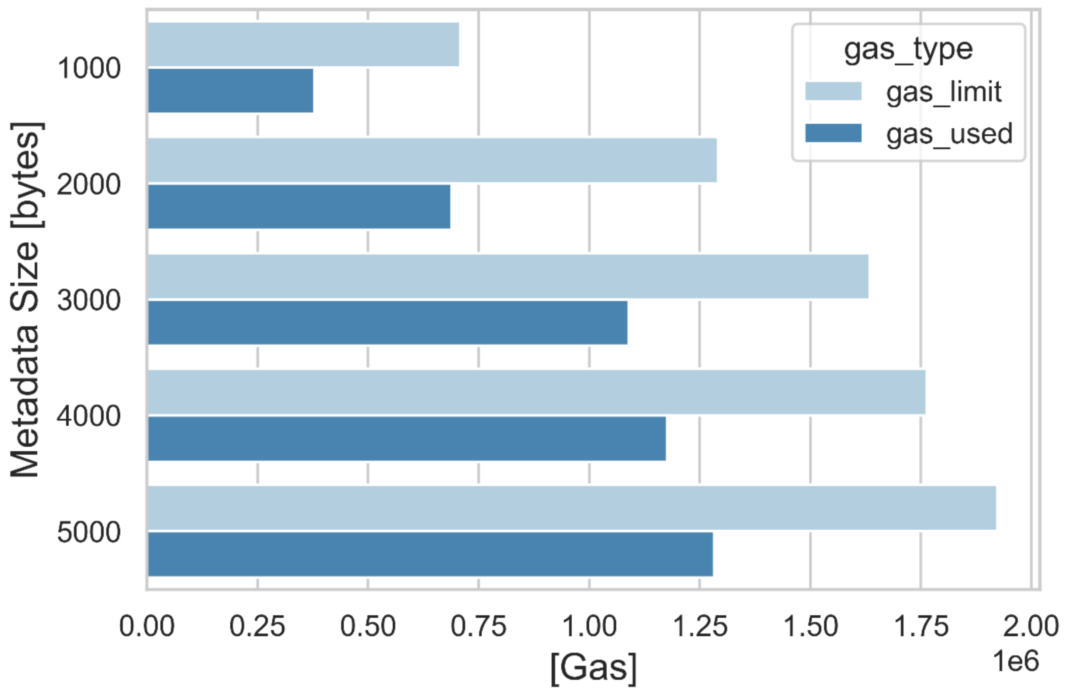


Figure 6.19 Gas Consumption on Adding New Metadata.

The implemented algorithms presented in 4.7 and 5.2.2 which is the complexity analysis are presented in Table 6.4 with the Big-O Notation technique. The addNewFirmware() is responsible for uploading metadata to the network and it takes metadata as an argument. The algorithm performs a comparison operation which is a constant operation taking O (1). If the comparison passes, then the metadata is inserted into a mapping structure. Mapping is essentially a kind of hash table where values are mapped to keys. Since the metadata is being inserted into the mapping, the operation will show a constant O (1). The last part will be to emit the event, which is a constant operation resulting in the time complexity of T(N) = 1 + 1 + 1 = 3. Hence, the total complexity is constantly at O (1). For checking firmware availability in isUpdateAvailable(), the algorithm first checks if the model is in the model list using the for-loop which executes N times, together with the operation inside its body. Therefore T(N) = N (for-loop) + 2N (2 times N comparison inside the loop) = 3N = N; hence, the order of growth is O (N). The order of growth for retrieveMetadata (), registerDev(), and getDevInfo() is affected by the for-loop which takes N steps O (N). The deleteDev(), updateDeviceInfo(), updateDeviceStatus(), and getDeviceStatus() consists of the instruction that retrieves the device status from the mapping, which makes the order of growth O (1).

Table 6.4 Algorithm Complexity Analysis

Algorithm	**Complexity**
addNewFirmware()	O (1)
isUpdateAvailable()	O (N)

retrieveMetadata ()	O (N)
registerDev()	O (N)
deleteDev()	O (1)
updateDevInfo()	O (1)
updateDeviceStatus()	O (1)
getDevInfo()	O (N)
getDeviceStatus()	O (1)

The efficient way of measuring the complexity of the algorithms is through gas. The gas price affects the execution time of the operation. Lowering the amount of gas price paid will lower the total cost of a given operation, it will also ensure that it takes longer. Paying a higher gas price will ensure a transaction is prioritized in the Blockchain, while, in most cases, paying a lower gas price will essentially ensure that a transaction will not take place for at least a few minutes. Higher gas prices generally mean that transactions will be completed faster, while lower gas prices mean they will take more time. The gas costs are shown in Table 6.4.

6.6 Requirements Satisfaction

This Section compares the proposed architecture and the state-of-the-art firmware updates mechanisms as shown in Table 6.5. It also provides how the properties and requirements were addressed by the proposed solution in Table 6.6. The Section explains how the proposed solution fulfilled each property with the requirements listed in Section 4.3.1.

Table 6.5 Comparison of The State-of-the-Art Against the Proposed Work

Authors	Low/Middle-End Device	High-End Device	Constrained-network	Availability	Confidentiality	Integrity	Authentication	Data Freshness	Performance Evaluation
Centralized-Based									
(Alexandre, 2016)		✓			✓	✓	✓		✓
(Pycom, 2018)	✓		✓			✓			
(Doddapaneni *et al.*, 2017)	✓		✓			✓			
(Reißmann and Pape, 2017)	✓					✓	✓		
(Lo and Hsu, 2019)			✓			✓	✓		
(Abdelfadeel *et al.*, 2020b)	✓		✓						✓
(Sahlmann *et al.*, 2021)	✓				✓	✓	✓	✓	✓
(Verderame *et al.*, 2021)	✓				✓	✓			✓
(Techniques, 2021)	✓		✓		✓	✓	✓		✓

(Charilaou *et al.*, 2021)	✓		✓			✓	✓		✓
Blockchain-Based									
(Lee and Lee, 2017)		✓		✓	✓	✓	✓		
(Yohan and Lo, 2019)		✓				✓			
(Mtetwa *et al.*, 2019)		✓		✓		✓			
(Witanto *et al.*, 2020)		✓		✓		✓	✓		✓
(Anastasiou *et al.*, 2020)				✓		✓	✓		✓
(Fukuda and Omote, 2021)		✓		✓	✓	✓	✓		✓
(Sanchez-gomez *et al.*, 2021)	✓		✓	✓	✓	✓	✓		
(Tsaur, Chang and Chen, 2022)		✓	✓						
Proposed Work	✓		✓	✓	✓	✓	✓	✓	✓

Table 6.6 Fulfilment of Requirements

Requirement Number	**Description**
REQ1	**Push Updates** – One of the requirements was to ensure that the system should enable administrators or device owners to schedule firmware updates to their devices to avoid network saturation and limit unintended downtime. The system met this requirement by implementing the python script that can be used in CLI to schedule firmware updates. The utilization of the script is shown in Figure 4.8. The Figure shows the command the owner of the devices uses to initiate the firmware update.
REQ2	**Manage Updates** – The implemented CLI script does not only limit the device owners to initiate the firmware update, but it also enables them to manage the updates of devices e.g., deletes, updates, and registers the low-powered devices to the Blockchain network. Hence, this fulfills the second requirement that there must be one component that manages updates of multiple microcontrollers that compose IoT devices. In addition, the FUS was implemented as a component that manages the entire firmware update process.
REQ3	**Over-The-Air Updates and Network Constrians Adoption** – The third requirement stated that the firmware update mechanism must adopt over-the-air updates, and the mechanism strategy should be adapted to the network bandwidth constraints. This study adopted the OTA mechanism instead of applying manual updates to low-powered devices.

	The mechanism adheres to the network restrictions, for instance, LoRaWAN is a restricted network with a very low data rate and requires a small amount of data to be exchanged. The proposed update mechanism ensured that the exchanged messages including the firmware image adhered to the LoRaWAN bandwidth. For example, if distributed firmware image is greater than the regionally specified bandwidth, the FUS will perform the firmware fragmentation based on the DR or the SF being utilized by the low-powered devices.
REQ4	• **Integrity** & **Authentication** – The integrity and authentication of data are provided by utilizing hashing, symmetric and asymmetric algorithms namely the SHA256, MAC, and ECDSA algorithms. ECDSA and SHA256 algorithms are used by the FUS to check firmware authenticity and integrity after it has been downloaded. The low-powered device also performs integrity and authenticity checks through the MAC algorithm based on the shared secret key. • **Confidentiality** – The proposed architecture provides confidentiality of data through AES encryption. The mode of operation for AES utilized by the architecture is the counter mode (MODE_CTR). The encryption of messages is performed in a different area, for example, the FUS encrypts the low-powered device information before it gets pushed to the public Blockchain network and decrypts the information when it retrieved data on the Blockchain network. The low-powered device also uses AES for the encryption and decryption of information such as session keys.
REQ5	**Availability** – The proposed architecture ensures the availability of firmware images through decentralized networks. The architecture utilizes the IPFS and Blockchain networks. Both networks rely on the set of nodes to achieve the high availability of firmware images. The traditional ways of storing the firmware are based on a central entity like a centralized server. The distributed and decentralized ways are good for achieving availability where the firmware is published on multiple

	devices and synced with the rest of the devices connected to the network. The architecture uses an infura node to store and retrieve the firmware image stored on the decentralized IPFS network. This allows access to the firmware image even if the manufacturer’s node is offline in the network. Just like the firmware image being stored on the decentralized IPFS network, the metadata is stored on the Blockchain network, which comprises many nodes that ensure the high availability of metadata. The architecture utilizes an infura node to connect and communicate the metadata with the rest of the Blockchain nodes.
REQ6	**Replay Attack** – Replay attack or Data freshness ensures that the messages are fresh and the man-in-the-middle has not replayed old messages. The data freshness in our architecture must be ensured because sensitive information like session keys and signatures are being exchanged hence, encryption alone is not sufficient. The architecture ensures the data freshness between the FUS and the low-powered. These two entities have a function that keeps track of a nonce value to ensure data freshness and the nonce values are encrypted and kept secret between the entities. For example, when the low-powered device receives the session key, it will decrypt them with the shared secret key and verify the nonces against the expected value of nonces. If the nonce values are correct, it will accept the respective message as fresh.
REQ7	**Low-power consumption** – The proposed architecture has been developed, tested, and evaluated in a constrained network. The constrained network being utilized is the LoRaWAN network which provides low-power consumption, and long-range connectivity, at low bandwidth between 250 bit/s and 11 kbit/s in Europe using LoRa modulation. The proposed architecture is demonstrated by utilizing constrained devices to show its suitability on a constrained network. The work also performs an evaluation of Blockchain operations, LoRaWAN, and cryptographic costs which are demonstrated in Section 5.2, Section 5.3, and Section 6.4.

Chapter 7: Conclusion

This chapter aims to summarise and conclude this study. The chapter summarizes the research problem and research questions introduced in Chapter 1, and how they were answered in this study. Finally, the recommendations for future work and the limitations are discussed.

7.1 Problem Summary

IoT consists of a massive number of devices and the devices will inevitably require patches to improve the performance after the deployment and functioning in the field. Most of the devices are deployed in environments with no Internet connectivity which makes it hard to reach and deliver firmware updates. Several studies have targeted securing and delivering firmware to different devices in different IoT networks trying to enhance the security of the devices. Studies based on constrained IoT networks specifically LoRaWAN as an emerging IoT technology, consider traditional ways of securing and delivering firmware updates to the devices. IoT devices are exponentially growing, and thousands of deployments are expected in the future. Thus, traditional approaches exhibit single-point-of-failure which is a downside to the IoT networks by looking at the way it scales. Hence, new ways of delivering firmware updates are required. Therefore, this study proposed to develop a decentralized solution as a better way to deliver firmware updates to constrained LoRaWAN networks using Blockchain technology.

7.2 Contributions

In this study, we reviewed the existing literature on firmware updates in IoT. Further, we identified the gaps in the current firmware update solution. We addressed these deficiencies by implementing an end-to-end firmware update mechanism for low-powered devices in the LoRaWAN network. The research makes the following contributions:

- This dissertation designed and implemented a Blockchain-based component called Firmware Update Service (FUS) which can be integrated with LoRaWAN application server and be responsible for handling firmware updates of the low-powered devices in the LoRaWAN network. This contribution comes from the fact that the currently existing research only conducts LoRaWAN firmware update research utilizing simulation software tools. Our study, therefore, provides the real implementation of firmware updates in LoRaWAN and integrates Blockchain technology to provide a resilient firmware update mechanism.

- The CLI tool that helps device owners or administrators to manage and schedule firmware updates to the low-powered devices in the LoRaWAN network was developed. To the best of our knowledge, the existing LoRaWAN research, that integrates Blockchain is based on the simulation tool and does not provide a way how to manage LoRa-end devices' firmware updates. Therefore, our study contributes by implementing the command line tool that can be utilized to schedule and manage the devices.
- And finally, the study evaluated the suitability of the proposed solution for low-powered devices in the LoRaWAN network.
- In addition, 4 scientific articles which include the publication of an article in an accredited journal and three conference proceedings articles proceeded from the research

7.3 Research Questions Answers

This dissertation wanted to answer the following main research question.

How can a Blockchain-based firmware update architecture for the LoRaWAN network be designed and implemented?

From the main research question, four sub-research questions were constructed. The questions and how each was answered are presented below.

1. **What is the “state of the art” in LoRaWAN firmware updates??**

To address this sub-question, it was essential to understand the existing firmware mechanism approaches and look at the tactics utilized to secure the firmware updates. As a result, the study looked at the different approaches in the literature review presented in Chapter 3. The approaches were categorized into two main categories: the centralized which is based on the client-server model, and the decentralized approach particularly based on Blockchain technology. In addition, each study was observed with the properties it aims to achieve. The properties include:

- Security properties - Security properties include confidentiality, integrity, authentication, and data freshness (replay attack).
- Type of IoT network – the IoT comprises of different types of networks. A certain study is either focused on the constrained network or not.

- Type of IoT device targeted – Section 2.1.1 described different devices that exist in the IoT networks, and the investigated studies were categorized based on the type of devices that were targeted. This includes the constrained low-end, middle-end, and high-end devices
- Performance Evaluation – certain studies only focused on proposing the firmware update solution design without providing the proof of concept, some only propose and provide the proof of concept without doing any evaluations. Therefore, the observed studies were investigated to see whether the performance evaluation was provided or not.

As a result, the identified properties in the literature helped to identify the gaps and directed our study to focus on constrained IoT devices in the constrained network while providing the required security attributes and evaluation.

2. **Why is Blockchain suitable for firmware updates in LoRaWAN?**

This research question was addressed using the background study which was observing the advantages of Blockchain technology and addressed by reviewing the literature in Chapter 2: In Section 3.1 different Blockchain LoRaWAN integration studies were reviewed to see how Blockchain is used in LoRaWAN. Section 3.1 showed that Blockchain technology can be suitable for LoRaWAN in many ways. For instance, LoRaWAN is based on symmetric cryptography and when Blockchain is integrated with LoRaWAN, it can take advantage of the advanced features of Blockchain in terms of security e.g., using asymmetric cryptography for enhancing LoRaWAN security. Moreover, when it comes to data that is generated by LoRa devices Blockchain provides a highly secured way of storing the data and tamper-proof data.

3. **How can a Blockchain-based firmware update mechanism that suits LoRaWAN be implemented?**

We tackled this question in Chapter 4: and Chapter 5: We focused on the design of the system architecture in Chapter 4: and the implementation of the system in Chapter 5: We identified the key requirements which must be met by firmware update mechanisms which in general are the IoT networks and also LoRaWAN. The requirements were obtained from the literature and the firmware updates recommendation from the report notes. These were presented in Section 4.3.1. For example, (Jongboom and Stokking, 2018a) came up with some recommendations that need to be followed when updating low-powered devices in LPWAN e.g. the mechanism must adhere to network restrictions. Thus, the design and implementation were based on such recommendations. The system architecture was presented in Section 4.4 and illustrated in Figure 4.3. The study implemented the independent components (that is FUS) that connect the integrated Ethereum Blockchain, LoRaWAN, and IPFS network. The FUS component is responsible for the entire firmware update process connected to LoRaWAN via the LoRaWAN application server. The Blockchain smart contracts were implemented using solidity and the developed smart contracts enforced the rules during the firmware updates and securely stored the low-powered device's information in the Blockchain.

4. **How can the proposed firmware update mechanism be evaluated?**

Experiments were conducted to evaluate the effectiveness of the proposed architecture. The experiments aimed to find out whether the architecture is suitable for constrained low-powered devices. This led us to examine certain metrics explained in Section 6.2 to determine the suitability of the architecture to the devices, for example examining memory consumption of cryptographic algorithms utilized to secure the firmware, etc. The architecture was evaluated using devices listed in Table 5.5 which include the LoRa node (LoPy), LoRa gateway (Raspberry Pi and RAK831 module), and the PC running LoRaWAN servers.

7.4 Summary

Privacy and security in the IoT are still in the early stages. Bugs and vulnerabilities were discovered on the devices while being active on the Internet, therefore, it is important to keep the security of the devices up to date to mitigate the vulnerabilities. This dissertation has presented the design and implementation of a Blockchain-based architecture targeting to delivery of firmware to low-powered constrained devices in the LoRaWAN network. The solution presented in this study showcases the potential of Blockchain technology in solving some of the issues found with centralized firmware update approaches. For example, since IoT devices are exponentially growing, the existing centralized solutions may not ensure the high availability of firmware thus, they exhibit single-point-of-failure which may seriously affect the security of the devices and the privacy of the consumers when security threats occur. Hence, the study took the advantage of the decentralized technology to enhance the security of firmware updates in LoRaWAN. The obtained results show that the architecture is suitable for LoRaWAN and constrained low-powered devices in the network. This suitability is observed through the examination of memory consumption during the update process. It depends on the firmware image size, for example, a larger firmware image size requires more resources during the verification process.

7.5 Limitations and Future Work

Even though the Blockchain-based solution for LoRaWAN is developed and presented, there are still many open issues that this work did not cover when it comes to firmware updates in constrained LoRaWAN networks. In this study, unfortunately, the larger-scale tests were not possible, due to the limited number of LoRa nodes we had. Furthermore, the scope of this study does not focus on the security of the bootloader. In future work, the study can be extended by improving the solution incorporating asymmetric cryptography for instance incorporating the Elliptic Curve Digital Signature Algorithm (ECDSA) and examining the effect of the Blockchain-based ECDSA algorithms during the device's verification of the firmware image.

References

Abdelfadeel, K. *et al.* (2020a) ‘How to make Firmware Updates over LoRaWAN Possible’, in *2020 IEEE 21st International Symposium on "A World of Wireless, Mobile and Multimedia Networks*, pp. 16–25. Available at: https://doi.org/10.1109/WoWMoM49955.2020.00018.

Abdelfadeel, K. *et al.* (2020b) ‘How to Make Firmware Updates over LoRaWAN Possible’, *Proceedings - 21st IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, WoWMoM 2020*, (February), pp. 16–25. Available at: https://doi.org/10.1109/WoWMoM49955.2020.00018.

Adam, I.O. and Dzang Alhassan, M. (2020) ‘Bridging the global digital divide through digital inclusion: the role of ICT access and ICT use’, *Transforming Government: People, Process and Policy*, 15(4), pp. 580–596. Available at: https://doi.org/10.1108/TG-06-2020-0114.

Alexandre, T. (2016) *UpdaThing : A secure and open firmware update system for Internet of Things devices*. Tecnico Lisboa. Available at: https://www.semanticscholar.org/paper/UpdaThing%3A-a-secure-firmware-update-system-for-of-Pinho/b8d685f27d6122cbd967a3db4bdf6d97fc46aa2d.

Alliance, L. *et al.* (2018) ‘LoRa Alliance Enhances LoRaWAN Protocol with New Specifications to Support Firmware Updates Over the Air’. Available at: https://resources.lora-alliance.org/technical-specifications/lora-alliance-enhances-lorawan-protocol-with-new-specifications-to-support-firmware-updates-over-the-air.

Alliance, L. (2018) ‘LoRaWAN 1.0.3 specification’, *Lora-Alliance.Org*, (1), pp. 1–72. Available at: https://lora-alliance.org/sites/default/files/2018-07/lorawan1.0.3.pdf.

Anastasiou, A. *et al.* (2020) ‘IoT Device Firmware Update over LoRa : The Blockchain Solution’, in *2020 16th International Conference on Distributed Computing in Sensor Systems (DCOSS) IoT*. IEEE, pp. 404–411. Available at: https://doi.org/10.1109/DCOSS49796.2020.00070.

Aqeel-ur-Rehman *et al.* (2016) ‘Security and privacy issues in IoT’, *International Journal of Communication Networks and Information Security*, 8(3), pp. 147–157. Available at: https://doi.org/10.4018/978-1-7998-8954-0.ch021.

Atzori, M. (2017) ‘Blockchain technology and decentralized governance: Is the state still necessary?’, *Journal of Governance and Regulation*, 6(1), pp. 45–62. Available at: https://doi.org/10.22495/jgr_v6_i1_p5.

Azam, N. *et al.* (2022) ‘Data Privacy Threat Modelling for Autonomous Systems: A Survey from the GDPR&#x0027;s Perspective’, *IEEE Transactions on Big Data*, pp. 1–27. Available at: https://doi.org/10.1109/TBDATA.2022.3227336.

Baranyi, P. *et al.* (2021) ‘Introducing the concept of internet of digital reality – part i’, *Acta Polytechnica Hungarica*, 18(7), pp. 225–240. Available at: https://doi.org/10.12700/APH.18.7.2021.7.12.

BBC (2021) *McDonald’s Hit by Data Breach in Taiwan and South Korea*. Available at:

https://www.bbc.com/news/business-57447404 (Accessed: 6 December 2021).

Bormann, Ersue and Keranen (2014) *Terminology for Constrained-Node Networks, Internet Engineering Task Force (IETF)*. Available at: http://www.springer.com/series/15440%0Apapers://ae99785b-2213-416d-aa7e-3a12880cc9b9/Paper/p18311.

Brtnik, V. (2018) *Master thesis Security Risk Assessment of LoRaWan*. Leiden University. Available at: https://openaccess.leidenuniv.nl/bitstream/handle/1887/64567/Brtnik_V_2018_CS.PDF?sequence=2.

Carstensen, A.K. and Bernhard, J. (2019) 'Design science research–a powerful tool for improving methods in engineering education research', *European Journal of Engineering Education*, 44(1–2), pp. 85–102. Available at: https://doi.org/10.1080/03043797.2018.1498459.

Charilaou, C. *et al.* (2021) 'Firmware update using multiple gateways in lorawan networks', *Sensors*, 21(19), pp. 1–19. Available at: https://doi.org/10.3390/s21196488.

Dale Liu *et al.* (2009) *Chapter 3 - An Introduction To Cryptography, Next Generation SSH2 Implementation*. Elsevier Inc. Available at: https://doi.org/B978-1-59749-283-6.00003-9.

Danish, S.M. *et al.* (2019) 'A Lightweight Blockchain based Two Factor Authentication Mechanism for LoRaWAN Join Procedure', *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 1–6. Available at: https://doi.org/10.1109/ICCW.2019.8756673.

Dika, A. and Nowostawski, M. (2017) 'Ethereum Smart Contracts: Security Vulnerabilities and Security Tools', (December). Available at: https://brage.bibsys.no/xmlui/bitstream/handle/11250/2479191/18400_FULLTEXT.pdf.

Doddapaneni, K. *et al.* (2017) 'Secure FoTA Object for IoT', *Proceedings - 2017 IEEE 42nd Conference on Local Computer Networks Workshops, LCN Workshops 2017*, pp. 154–159. Available at: https://doi.org/10.1109/LCN.Workshops.2017.78.

Dooley, K. (2002) 'Simulation Research Methods Kevin Dooley Arizona State University', (April), pp. 829–848.

Durand, A., Gremaud, P. and Pasquier, J. (2018) 'Resilient, crowd-sourced LPWAN infrastructure using blockchain', *CRYBLOCK 2018 - Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems, Part of MobiSys 2018*, pp. 25–29. Available at: https://doi.org/10.1145/3211933.3211938.

European Data Protection Supervisor (2019) *Introduction to the Hash Function as a Personal Data*. Available at: https://edps.europa.eu/sites/edp/files/publication/19-10-30_aepd-edps_paper_hash_final_en.pdf.

Fatjon Muca (2014) 'Reaserach Methods', in. Available at: https://www.academia.edu/29798644/Method_Research.

Fukuda, T. and Omote, K. (2021) 'Efficient Blockchain-based IoT Firmware Update

Considering Distribution Incentives', *2021 IEEE Conference on Dependable and Secure Computing, DSC 2021* [Preprint]. Available at: https://doi.org/10.1109/DSC49826.2021.9346265.

Gambiroza, J.C. *et al.* (2019) 'Capacity in lorawan networks: Challenges and opportunities', *2019 4th International Conference on Smart and Sustainable Technologies, SpliTech 2019* [Preprint], (June). Available at: https://doi.org/10.23919/SpliTech.2019.8783184.

George Corser *et al.* (2017) 'Internet of Things (Iot) Security Best Practices', *Ieee*, 1(February), p. 17. Available at: https://internetinitiative.ieee.org/images/files/resources/white_papers/internet_of_things_feb2017.pdf.

Go Ethereum (2021) *Official Go implementation of the Ethereum Protocol*. Available at: https://geth.ethereum.org/ (Accessed: 6 December 2021).

Greenberg, A. (2020) *This Bluetooth Attack Can Steal a Tesla Model X in Minutes*. Available at: https://www.wired.com/story/tesla-model-x-hack-bluetooth/ (Accessed: 14 August 2022).

Hackernoon (2021) *IPFS: A Beginner's Guide*. Available at: https://hackernoon.com/a-beginners-guide-to-ipfs-20673fedd3f (Accessed: 6 December 2021).

Infura (2021) *Ethereum | Infura Documentation*. Available at: https://infura.io/docs/ethereum#section/Make-Requests/JSON-RPC-Methods (Accessed: 6 December 2021).

IPFS (2021a) *Distributed Hash Tables*. Available at: https://docs.ipfs.io/concepts/dht (Accessed: 6 December 2021).

IPFS (2021b) *How Bitswap works*. Available at: https://docs.ipfs.io/concepts/bitswap/ (Accessed: 6 December 2021).

IT Governance UK (2021) *Data breaches and cyber attacks quarterly review: Q3 2021 - IT Governance UK Blog*. Available at: https://www.itgovernance.co.uk/blog/data-breaches-and-cyber-attacks-quarterly-review-q3-2021 (Accessed: 6 December 2021).

Johnson, S.D. *et al.* (2020) 'The impact of IoT security labelling on consumer product choice and willingness to pay', *PLoS ONE*, 15(1), pp. 1–21. Available at: https://doi.org/10.1371/journal.pone.0227800.

Jongboom, J. and Stokking, J. (2018a) 'Enabling firmware updates over LPWANs', *Embedded World Conference* [Preprint]. Available at: http://janjongboom.com/downloads/ew2018-paper.pdf.

Jongboom, J. and Stokking, J. (2018b) *Enabling firmware updates over LPWANs, Embedded World 2018 Exhibition & Conference*. Embedded World. Available at: http://janjongboom.com/downloads/ew2018-paper.pdf.

Kaliski, B. (2011) 'Hard-Core Bit', *Encyclopedia of Cryptography and Security*, 47(3), pp. 534–535. Available at: https://doi.org/10.1007/978-1-4419-5906-5_412.

Kvarda, L. *et al.* (2016) 'Software implementation of a secure firmware update solution in an

IOT context', *Advances in Electrical and Electronic Engineering*, 14(4Special Issue), pp. 389–396. Available at: https://doi.org/10.15598/aeee.v14i4.1858.

Lee, B. and Lee, J.H. (2017) 'Blockchain-based secure firmware update for embedded devices in an Internet of Things environment', *Journal of Supercomputing*, 73(3), pp. 1152–1167. Available at: https://doi.org/10.1007/s11227-016-1870-0.

Leverege LCC (2018) *An Introduction to the Internet of Things*. 1st edn. Leverege LCC. Available at: https://www.leverege.com/iot-ebook/introduction.

Lin, J., Shen, Z. and Miao, C. (2017) 'Using Blockchain Technology to Build Trust in Sharing LoRaWAN IoT', *Proceedings of the 2nd International Conference on Crowd Science and Engineering - ICCSE'17*, (February), pp. 38–43. Available at: https://doi.org/10.1145/3126973.3126980.

Lo, N.-W. and Hsu, S.-H. (2019) *A Secure IoT Firmware Update Framework Based on MQTT Protocol*. Available at: https://doi/10.1007/978-3-030-30443-0.

Makhdoom, I. *et al.* (2019) 'Journal of Network and Computer Applications Blockchain ' s adoption in IoT : The challenges , and a way forward', *Journal of Network and Computer Applications*, 125(November 2018), pp. 251–279. Available at: https://doi.org/10.1016/j.jnca.2018.10.019.

Manoj Athreya, A. *et al.* (2021) 'Peer-to-Peer Distributed Storage Using InterPlanetary File System', in N.N. Chiplunkar and T. Fukao (eds) *Advances in Artificial Intelligence and Data Engineering*. Singapore: Springer Singapore, pp. 711–721. Available at: https://doi.org/10.1007/978-981-15-3514-7_54.

Marais, J.M., Abu-Mahfouz, A.M. and Hancke, G.P. (2020) 'A survey on the viability of confirmed traffic in a LoRaWAN', *IEEE Access*, 8, pp. 9296–9311. Available at: https://doi.org/10.1109/ACCESS.2020.2964909.

Miller, C. and Valasek, C. (2015) 'Remote Exploitation of an Unaltered Passenger Vehicle', *Defcon 23*, 2015, pp. 1–91. Available at: http://illmatics.com/Remote Car Hacking.pdf.

Mtetwa, N. *et al.* (2019) 'Secure Firmware Updates in the Internet of Things : A survey', in *International Multidisciplinary Information Technology and Engineering Conference*, pp. 1–7. Available at: https://doi.org/10.1109/Cybermatics_2018.2018.00051.

Mtetwa, N., Tarwireyi, P. and Adigun, M. (2019) 'Secure the Internet of Things Software Updates with Ethereum Blockchain', in *Proceedings - 2019 International Multidisciplinary Information Technology and Engineering Conference, IMITEC 2019*. IEEE, pp. 1–6. Available at: https://doi.org/10.1109/IMITEC45504.2019.9015865.

Mtetwa, N.S. *et al.* (2019) 'Secure Firmware Updates in the Internet of Things: A survey', in *Proceedings - 2019 International Multidisciplinary Information Technology and Engineering Conference, IMITEC 2019*. IEEE, pp. 1–7. Available at: https://doi.org/10.1109/IMITEC45504.2019.9015845.

NetScouts (2018) 'Dawn of the TerrorBIT Era NETSCOUT Threat Intelligence Report-Powered by ATLAS Findings from Second Half 2018', p. 2. Available at:

https://www.netscout.com/sites/default/files/2019-02/SECR_001_EN-1901 - NETSCOUT Threat Intelligence Report 2H 2018.pdf.

Ojo, M.O. *et al.* (2018) 'A Review of Low-End, Middle-End, and High-End Iot Devices', *IEEE Access*, 6(November), pp. 70528–70554. Available at: https://doi.org/10.1109/ACCESS.2018.2879615.

OWASP (2018) 'OWASP Top 10 Internet of Things', *Salem Press Encyclopedia of Science*, pp. 5–7. Available at: http://search.ebscohost.com/login.aspx?direct=true&db=ers&AN=100558386&site=eds-live.

Ozyilmaz, K.R. and Yurdakul, A. (2019) 'Designing a Blockchain-Based IoT with Ethereum, Swarm, and LoRa: The Software Solution to Create High Availability with Minimal Security Risks', *IEEE Consumer Electronics Magazine*, 8(2), pp. 28–34. Available at: https://doi.org/10.1109/MCE.2018.2880806.

PandaSecurity (2021) *The Stolen Source Code For FIFA 21 Was Just Published Online*. Available at: https://www.pandasecurity.com/en/mediacenter/security/source-code-fifa-21/ (Accessed: 6 December 2021).

Pycom (2018) *Pycom Documentation*. Available at: https://docs.pycom.io/chapter/datasheets/.

Regenscheid, A. (2018) 'Platform Firmware Resiliency Guidelines', *NIST Special Publication*, 193(May). Available at: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-193.pdf.

Rehman, M.M.U., Rehman, H.Z.U. and Khan, Z.H. (2020) 'Cyber-attacks on medical implants: A case study of cardiac pacemaker vulnerability', *International Journal of Computing and Digital Systems*, 9(6), pp. 1229–1235. Available at: https://doi.org/10.12785/ijcds/0906020.

Reißmann, S. and Pape, C. (2017) 'An Over the Air Update Mechanism for ESP8266 Microcontrollers', *ICSNC 2017 : The Twelfth International Conference on Systems and Networks Communications An*, (October), pp. 11–17. Available at: https://www.researchgate.net/publication/320335879_An_Over_the_Air_Update_Mechanism_for_ESP8266_Microcontrollers.

Sahlmann, K. *et al.* (2021) 'Mup: Simplifying secure over-the-air update with mqtt for constrained iot devices', *Sensors (Switzerland)*, 21(1), pp. 1–21. Available at: https://doi.org/10.3390/s21010010.

Sanchez-gomez, J. *et al.* (2021) 'Holistic IoT Architecture for Secure Lightweight Communication , Firmware Update , and Trust Monitoring', pp. 353–358. Available at: https://doi.org/10.1109/SmartIoT52359.2021.00066.

Schiller, E. *et al.* (2022) 'Landscape of IoT security', *Computer Science Review*, 44, p. 100467. Available at: https://doi.org/10.1016/j.cosrev.2022.100467.

Sivagami, P. *et al.* (2021) 'IoT Ecosystem- A survey on Classification of IoT'. Available at: https://doi.org/10.4108/eai.16-5-2020.2304170.

Stevens, M. *et al.* (2017) 'The first collision for full SHA-1', *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10401 LNCS(July), pp. 570–596. Available at: https://doi.org/10.1007/978-3-319-63688-7_19.

Subodhnarayan, S. *et al.* (2018) *Design and Implementation of an Integrated Water Quality Monitoring System and Blockchains*. University of Zurich. Available at: https://files.ifi.uzh.ch/CSG/staff/Rafati/Sanjiv-Jha-MA.pdf.

Sun, Y. *et al.* (2014) 'Data Security and Privacy in Cloud Computing', *International Journal of Distributed Sensor Networks*, 2014. Available at: https://doi.org/10.1155/2014/190903.

Swanson, M. (2020) 'LoRaWAN: Firmware Updates Over-the-Air', in, pp. 217–225. Available at: https://doi.org/10.1007/978-3-319-64933-7_10.

Tan, M., Sun, D. and Li, X. (2021) 'A Secure and Efficient Blockchain-based Key Management Scheme for LoRaWAN', in. Available at: https://doi.org/10.1109/WCNC49053.2021.9417304.

Techniques, R. (2021) 'Secure LoRa Firmware Update with Adaptive Data Rate Techniques', 21(7), pp. 1–17. Available at: https://doi.org/https://doi.org/10.3390/s21072384.

TexasInstruments (2015) 'Secure In-Field Firmware Updates for MSP MCUs', in, pp. 1–13. Available at: https://www.ti.com/lit/slaa682.

The Things (2021). Available at: https://www.thethingsnetwork.org/docs/lorawan/what-is-lorawan/ (Accessed: 6 December 2021).

Truffle (2021) *Ganache: A Tool for Creating a Local Blockchain for Fast Ethereum Development.* Available at: https://github.com/trufflesuite/ganache (Accessed: 6 December 2021).

Tsaur, W., Chang, J. and Chen, C. (2022) 'A Highly Secure IoT Firmware Update Mechanism Using Blockchain'. Available at: https://doi.org/10.3390/s22020530.

University of Delaware (2021) *Managing data confidentiality*. Available at: https://www1.udel.edu/security/data/confidentiality.html (Accessed: 1 February 2022).

Vahdati, Z. *et al.* (2019) 'Comparison of ECC And RSA Algorithms in IoT', 97(16). Available at: https://www.semanticscholar.org/paper/COMPARISON-OF-ECC-AND-RSA-ALGORITHMS-IN-IOT-DEVICES-Vahdati/a43da6aa57ca8dbeeeb70c144a25b0f288caa8bb.

Verderame, L., Ruggia, A. and Merlo, A. (2021) 'PATRIOT: Anti-Repackaging for IoT Firmware', pp. 1–9. Available at: http://arxiv.org/abs/2109.04337.

Vlachos, I. and Hatziargyriou, N. (2019) 'Design and Implementation of a Decentralized Amr System Using Blockchains , Smart Contracts and Lorawan', (June), pp. 3–6.

Votipka, D. *et al.* (2020) 'Understanding security mistakes developers make: Qualitative analysis from build it, break it, fix it', in *Proceedings of the 29th USENIX Security Symposium*, pp. 109–126. Available at: https://dl.acm.org/doi/10.5555/3489212.3489219.

Vujičić, D., Jagodić, D. and Randić, S. (2018) ‘Blockchain technology, bitcoin, and Ethereum: A brief overview’, *2018 17th International Symposium on INFOTEH-JAHORINA, INFOTEH 2018 - Proceedings*, 2018-Janua, pp. 1–6. Available at: https://doi.org/10.1109/INFOTEH.2018.8345547.

Witanto, E.N. *et al.* (2020) ‘A blockchain-based ocf firmware update for IoT devices’, *Applied Sciences (Switzerland)*, 10(19), pp. 1–22. Available at: https://doi.org/10.3390/app10196744.

Yohan, A. and Lo, N.W. (2019) ‘An Over-The-Blockchain Firmware Update Framework for IoT Devices’, *DSC 2018 - 2018 IEEE Conference on Dependable and Secure Computing*, pp. 1–8. Available at: https://doi.org/10.1109/DESEC.2018.8625164.

Zandberg, K. *et al.* (2019a) ‘Secure Firmware Updates for Constrained IoT Devices Using Open Standards: A Reality Check’, *IEEE Access*, 7, pp. 71907–71920. Available at: https://doi.org/10.1109/ACCESS.2019.2919760.

Zandberg, K. *et al.* (2019b) ‘Secure Firmware Updates for Constrained IoT Devices Using Open Standards: A Reality Check’, *IEEE Access*, 7, pp. 71907–71920. Available at: https://doi.org/10.1109/ACCESS.2019.2919760.

Zarrin, J. *et al.* (2021) ‘Blockchain for decentralization of internet: prospects, trends, and challenges’, *Cluster Computing*, 24(4), pp. 2841–2866. Available at: https://doi.org/10.1007/s10586-021-03301-8.

Appendix A: Code for Adding Metadata on Blockchain

```
function addNewFirmware(string memory_model, string memory _version, string memory _f_metadata
    public{

    require(msg.sender==manufacturerID, "Not authorized for such operation.");
    for(i = 0; i<modelList.length; i++){
        if(sha256(bytes(_model))==sha256(bytes(modelList[i]))){
            exist = true;
        }
    }
    ...

    // Event
    emit addNewMetadata(_model, _version, _f_metadata);
}
```

Appendix B: Code for Checking Latest Firmware Update

```
function isUpdateAvailable(string memory _model,  string memory _f_version) public view
    returns(bool){

    for(i = 0; i<modelList.length; i++){
        if(sha256(bytes(_model))==sha256(bytes(modelList[i]))){
            if(sha256(bytes(metadataList[_model].f_version))>
                sha256(bytes(_f_version))){
                return true;
                ...
```

Appendix C: Code for Retrieving Metadata

```
function getMetadata(string memory _model) public view returns(string memory){
        return metadataList[_model].f_metadata;
}
```

Appendix D: Partial Code Snippet for Device Registration

```
function registerDev(string memory _appID, string memory _devID, string memory _s_address,
    string memory _w_address, string memory _model, string memory _version) public
    returns(bool){
    ...

    require(msg.sender==FUSID, "Not Authorized for such operation.");

    ...
}
```

Appendix E: Deleting the End Device

```
function deleteDev(string memory _devID) public{
    require(msg.sender==FUSID, "Not Authorized for such operation.");
    if(devices[_devID].exist){
        delete devices[_devID];
        noOfDevs--;
    }
}
```

Appendix F: Partial Code Snippet for Updating Device Information

```
function updateDeviceInfo(string memory _appID, string memory _devID, string memory _s_address,
    string memory _w_address, string memory model, string memory _version, string _status)
    public returns(bool){
        require(msg.sender ==FUSID, "Not Authorized for such operation.");

        ...
```

Appendix G: Code Snippet for Updating Device Status

```
function updateDeviceStatus(string memory _devID, string memory _status) public{
    require(msg.sender==FUSID, "Not Authorized for such operation.");
    devices[_devID].status = _status;
}
```

Appendix H: Code for Get the Device Information

```
function getDevInfo(string memory _devID) public view returns(string memory, string memory,
    string  memory, string memory, string memory, string memory){

    return(devices[_devID].appID, devices[_devID].s_address,devices[_devID].w_address,
        devices[_devID].model, devices[_devID].version,  devices[_devID].status
    );
}
```