OTA Firmware Updates for LoRaWAN Using Blockchain

Njabulo S. Mtetwa
Department of Computer Science
University of Zululand
KwaDlangezwa 3886, South Africa
mthethwansm@gmail.com

Nombuso Sibeko
Department of Computer Science
University of Zululand
KwaDlangezwa 3886, South Africa
nombuson@gmail.com

Paul Tarwireyi
Department of Computer Science
University of Zululand
KwaDlangezwa 3886, South Africa
tarwireyip@unizulu.ac.za

Adnan M. Abu-Mahfouz
Council for Scientific and Industrial Research (CSIR)
Pretoria 0184, South Africa
a.abumahfouz@ieee.org

Abstract— **LoRaWAN is one of the LPWAN technologies that has become popular in both industries and research. LoRaWAN aims to provide long-range communication and empowers low-powered devices to last for years in the field. LoRaWAN relies on the symmetric cryptography to provide end-to-end encryption. Even though LoRaWAN relies on symmetric cryptography, there are recent works that try to enhance security of LoRaWAN by incorporating technologies like Blockchain. Blockchain is a decentralized peer-to-peer network that provides tamperproof and immutability of data. This paper proposes a Blockchain-based firmware update mechanism to enhance firmware update in LoRaWAN as well as managing the update process. This mechanism aims to provide updates by ensuring authenticity, and integrity of the firmware. The mechanism focuses more on devices that are too constrained in resources, hence for that purpose we evaluated the cost involved in some cryptographic operations taken to ensure security during firmware updates. We conclude that the approach is feasible for constrained devices in LoRaWAN network by evaluating the memory usage of the cryptographic operation used by the end device.**

Keywords— **LoRaWAN, Security, Firmware, Blockchain, Resource-constrained devices, Internet of Things, LPWAN**

I. INTRODUCTION

The Internet of Things (IoT) has been growing exponentially [1] in the past few years and more devices are still expected in the future. Several communication technologies such as Wi-Fi, Cellular, and Low-Power Wide Area Network (LPWAN) are used to establish a connection between the vast of connected devices. LoRa in conjunction with LoRaWAN are mostly used and popular LPWAN technology with the aim of providing long-range communication and low-power consumption [2]. LoRa refers to a physical layer utilizing a chirp spread spectrum technique to provide long-range connectivity whereas LoRaWAN is a protocol that builds on top of LoRa and creates the network layer for managing traffic.

Most of the IoT devices were built with no security in mind hence; they are vulnerable to some attacks. Vulnerabilities enable hackers to gain access to the devices illegally, which can bring harm to the entities that rely on the IoT devices. For instance, one of the top hacks took place in 2015 called the jeep hack [3]. This hack was performed by two researchers who took advantage of many vulnerabilities including the firmware update vulnerability and reverse engineered the firmware to retrieve sensitive information such as, encryption algorithms, sensitive URLs, API and encryption keys. The researchers were able to take control of a jeep using the vehicle's Controller Area Network (CAN) bus that enables communication between different elements on vehicle such as steering wheel, breaks, heaters, locks, headlights etc. The CAN messages were sent to take control of various elements of the vehicle to make it speed up, slow down and even veer off the road. The lack of security on firmware update made this attack possible therefore, reliable mechanisms are required to ensure security during the firmware updates. Open Web Application Security Project (OWASP) listed vulnerabilities that attackers use to compromise the IoT devices [4]. The attacks include insufficient authentication authorization, lack of transport encryption, privacy concerns, inadequate security configuration, insufficient physical security and insecure firmware updates. Attackers use these vulnerabilities to illegally gain access to devices and perform nefarious activities such as retrieving sensitive user information like passwords, encryptions keys etc.

LoRaWAN provides security of the LoRa end devices through symmetric-key cryptography. Advanced Encryption Standard (AES) is utilized as an encryption algorithm to provide end-to-end encryption between LoRa devices and the servers. Despite the fact that LoRaWAN guarantees security through symmetric cryptography the replay attack can occur if the network does not appraise it [5]. Recently, some works have taken advantage of integrating LoRaWAN with other technologies as a means of adding extra layer of security. One of these technologies is Blockchain technology. Blockchain technology is a decentralized peer-to-peer network that is not managed by a third party [6]. Blockchain consists of many nodes that validates a group of transaction where each node saves a copy of a Blockchain ledger. This has an advantage since the data is distributed and is not controlled by one entity or the third party. One can deploy a smart contract, which cannot be updated, or manipulated by any entity; hence, Blockchain enables data to be immutable and tamper-proof. Asymmetric cryptography and hashing is a security behind the Blockchain.

A study by [7] integrated LoRaWAN with Blockchain to improve security in a join procedure since it is subject to replay attack. The proposed mechanism utilized Ethereum Blockchain where the gateway and network server are connected to Blockchain via an agent node through an interface.

In [8], the authors propose a Blockchain based solution to build an open, trusted decentralized, tamper-proof system for LoRaWAN. In the proposed system, Blockchain was integrated on the network server, however the system was only proposed then in [9] came up with a proof of concept by implementing the forwarding network server and integrated Blockchain with it. The aim of these integrations is to strengthen the security of LoRaWAN and to mitigate the vulnerabilities in IoT. Most of these vulnerabilities are found, after the devices are deployed and operating in the field. This means that the device manufacturers are expected to release new firmware versions to fix bugs, to improve device functionality and all this has to be done securely.

In this paper, we propose and implement the mechanism on how Blockchain can be utilized to provide security in firmware updates to LoRa end devices. We propose a mechanism that can be suitable for the devices that are constrained in resources. Blockchain acts as an extra layer of security to determine integrity and authenticity of the firmware on top of LoRaWAN. The proposed system is validated using the LoPy and expansion board 2.0 that is ESP32 chip based. The Ethereum test network called rinkeby is used as a public Blockchain network together with the InterPlanentary File System (IPFS) for storing the firmware image [10]. For the transmission of LoRa packets, we utilize the Things Network stack v3.8.3. The Preliminary results show that the approach is feasible for constrained devices in LoRaWAN network.

II. Related Work

The LoRa Alliance [11] has come up with the new specification to make the firmware update more easily in LoRaWAN. These specifications include multicast, fragmentation, clock synchronisation. There is not much work based on providing firmware updates in LoRaWAN, therefore we also look on the few ones using LoRaWAN and how other constrained devices are being update on the Internet of Things. Recently, the authors of [12] implemented a simulation tool called FUOTASim to show how firmware update can be achieved in LoRaWAN. The work also shows how the large number of LoRa devices can be updated using new LoRa specifications. The security during firmware update was out of the scope for this work. In [13] the authors propose how the LoRa devices could be updated; however, the method uses the traditional network technology to convey firmware image to the end device. This is not considered as a good solution for the devices operating on battery; hence, it may consume more power during the firmware updates and not suitable for the devices that are deployed in certain areas with no internet connection.

There are also Blockchain-based firmware updates mechanisms that target constrained devices. The authors [14] proposed a scheme that utilizes a Blockchain technology to securely check the firmware version, validate the correctness of the firmware and download the latest firmware for the embedded devices. In the proposed scheme, every IoT device represented a node in the Blockchain, which means they are required to store the Blockchain ledger in their local storage. The challenge with this is that most of the IoT devices have limited resources such as energy, computation and storage capacity. This mechanism might be difficult to be implemented in the real-world IoT environment.

The authors of [15] propose and implemented a decentralized firmware update framework called Código network, which was implemented on top of the Ethereum Blockchain, and the IPFS network. The target was to achieve a framework, which will allow no single point of failure, scalability, transparency of firmware updates, equivalent security code with code signing. The code signing, which was achieved through the use of digital signature and also through the use of Ethereum smart contract checked whether the firmware has been corrupted. This was achieved by comparing the hash stored in the smart contract with the hash of the firmware. The proposed solution was experimented with 10MB of the firmware image. The firmware was distributed in three different storages including the central server, BitTorrent and IPFS. The proposed mechanism is completely not suitable on the constraint networks and for constrained device such as class 0, class 1 since; it consists of the firmware image with large size of 10MB.

In this paper, we to provide a Blockchain based firmware update mechanism for LoRaWAN considering the security constrained devices.

III. Proposed Architecture

In this section, we start by listing the requirements of the propose system and then explain the proposed architecture illustrated in Fig. 1. The architecture has two subsystems. LoRaWAN and independent Blockchain network.

A. Proposed Architecture Requirements

The goal of the proposed architecture is to.

1) Suitable for resource constrained devices: The integration of Blockchain and LoRaWAN must suit LoRa devices which fall in class-0 and class-1 (constrained devices).

2) No utilization of any traditional wireless technology: The proposed architecture must only rely on LoRa and LoRaWAN to deliver the firmware to the end device. This ensures the end device does not consume lot of energy during the firmware update process.

3) Authenticity and Integrity: This is an important part of the firmware updates. The authenticity and the integrity of the firmware must be achieved.

4) No single point of failure: The update mechanism must be able to get the requested firmware image even if the manufacturer's node/repository is not available.

5) Multi-vendor/heterogeneous IoT devices: LoRaWAN consists of different devices from different manufacturers, therefore, the update mechanism must be design to support heterogeneous IoT device network with multiple manufacturers.

B. LoRaWAN

LoRaWAN consists of four components namely, LoRa end device, LoRa gateway, network server, application server and update service.

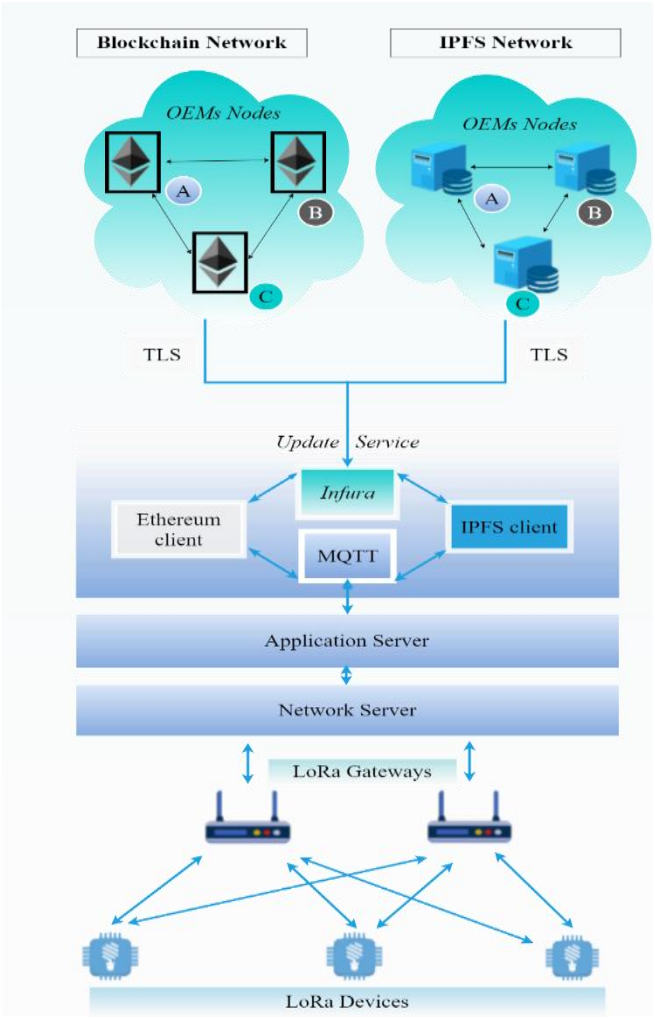

Fig 1. System architecture.

1) LoRa end device: An entity that needs firmware updates. These devices are very constrained with low processing; storage capabilities hence, it becomes difficult to incorporate advance cryptographic techniques. They are low-powered and most of the time they wake up, send data to the gateway then go back to sleep. In this proposed architecture, Blockchain is not integrated on the end devices because of the resources they have.

2) LoRa gateway: The gateway receives LoRa packet from the LoRa end devices and transmit packet to at least one or more network servers attached to it.

3) Network Server: The network server is responsible for handling du-duplication of packets from multiple gateways, handles devices join requests, queuing the downlink messages and send sends them to the gateway. The network session key (NkwSKey) is used for integrity validation of messages between the end device and the network server

4) Application Server: Uses application session key (AppSKey) to encrypt and decrypt the payloads. It has of multiple integrations such as HTTP, Message Queuing Telemetry Transport (MQTT), etc. MQTT integration is used to connect the application server and the update service. The interception of possible Blockchain network and LoRaWAN is based possible through this integration

5) Update Service: Update service is connected with application server via MQTT protocol. The entire update process is controlled by the update service which does these number of tasks:

- Subscribes and listens to registered devices MQTT topics from the application server.
- Handles the device firmware requests.
- Connects to the InterPlanentary File System (IPFS), Blockchain network and continuously listens for any new incoming firmware update event on the Blockchain.
- Get the latest firmware from the IPFS and performs the fragmentation based on the SF or data rates used by the end device.
- Performs cryptographic operations such as generating the session keys to be used by the end device during that particular session of firmware update, encrypt confidential data i.e. integrity hash, moreover it determines the authenticity and the integrity of the firmware image before, the image is sent over LoRaWAN.
- Update the state of the end device to the Blockchain.

C. Blockchain and IPFS

The manufacturers share their device firmware image shared publicly. This enables the device owners to download the image and update their devices. In our proposed mechanism, the public Blockchain is used to publicly share the firmware information like metadata. The firmware image is public stored on the decentralized IPFS, which enable high availability of the firmware. Manufacturer of the devices needs to own a blockchain node that synchronizes with the network together with the IPFS node, which is connected to the IPFS network. In our mechanism, we used the infura service nodes for both IPFS and blockchain instead of running our local nodes.

IV. Firmware Update Interaction

This section mainly explains the interaction between the components involved during the firmware update process. We start by describing the assumptions of the proposed system and then explain the interaction, which is classified in four main phases: device registration, firmware upload, firmware initiation, and finally, the entire update process.

A. Assumptions of the proposed architecture

The proposed system has the number of following assumptions:

1) The firmware update is applied on constrained devices with low processing capabilities to perform heavy cryptographic operations.

2) The firmware is stored on the open repository and can be downloaded by anyone; therefore, there is no need to encrypt the firmware from our update service to the end device. However, the content is also shared via SSL between application server and update service.

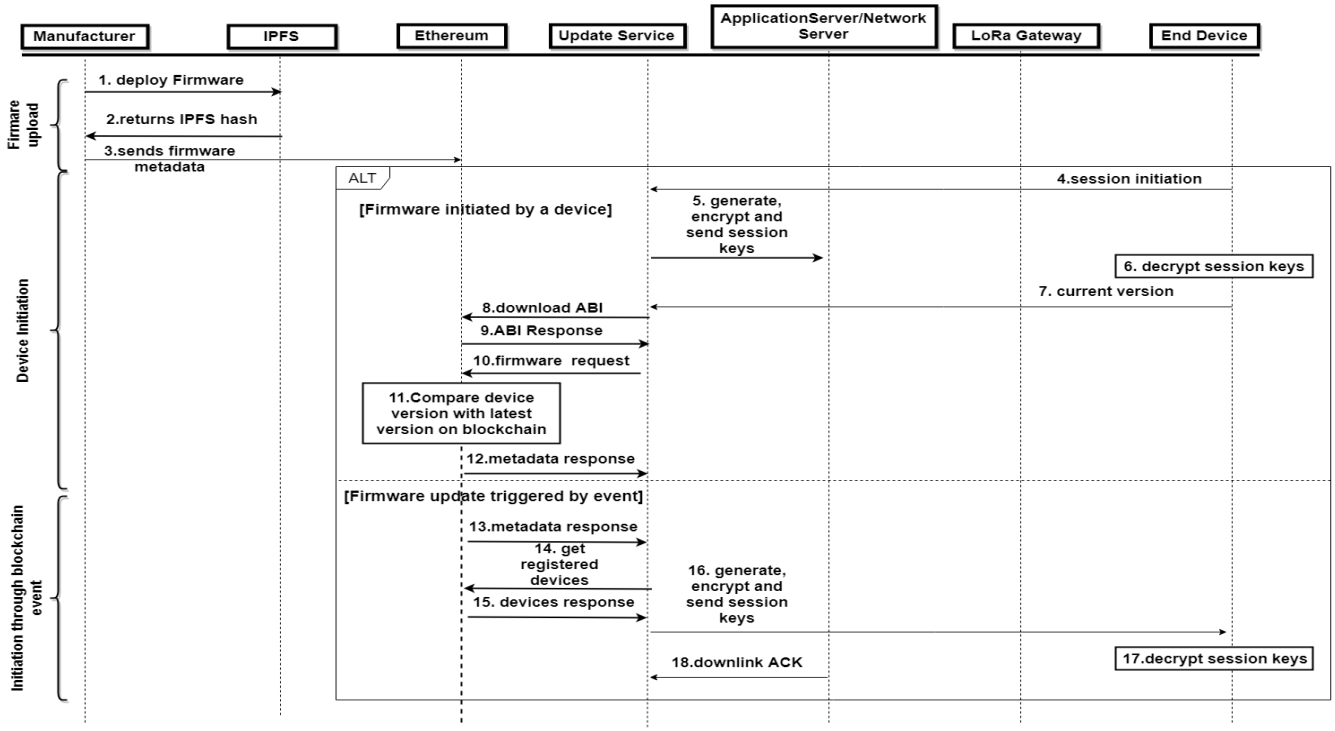

Fig 2. Interaction of system components.

Moreover, encrypting each firmware fragment will result in maximizing the packet size. For instance, if AES-128 is used in secure modes like CFB/CRT mode the 128 bits (16 bytes) of extra payload for the initialization vector is required.

3) Encryption and decryption keys are stored in a device secure module.

B. Device Registration

In order for the end device to receive the firmware updates, the registration to the update service is required. The update service needs the smart contract address of the manufacturer and the wallet address of the manufacturer and the model. This information is saved encrypted by the update service on the Blockchain network. A smart contract address will be used by the update service to locate an ABI of the manufacturer of the device and then use that ABI to interact with the smart contract on the Ethereum Blockchain. The update service generates the update key (UDTKey) of 128 bits which acts as a master key. UDTKey is used for encryption, decryption of session keys.

C. Firmware Upload

The new firmware image must be uploaded to the public repository. Decentralized storage IPFS is used to ensure high availability of the firmware image, so that the update service gets the firmware even if the manufacturer's repository is not available. At this point, the manufacturer creates smart contracts and is deployed on the Ethereum network. The firmware image is sent along with the firmware metadata. Firmware metadata contains information such as the size, devices the firmware is targeting, location of the firmware image, the integrity hash of the firmware, the signature of the manufacturer etc. The purpose of the firmware metadata is to ensure that the device installs the right firmware from the right manufacturer. It is recommended to store metadata on Blockchain since Blockchain is immutable and provides tamper-proof. The manufacturer signs the metadata with the private key and produces a signature that is sent along with the metadata. Ethereum Blockchain signature is based on the Elliptic Curve Digital Signature Algorithm (ECDSA) with 72 hexadecimal characters (36 bytes) which is quite different from RSA signature.

D. Firmware Initiation

This section shows how firmware update is initiated. The firmware update can be started based on two events. The first is when the device joins a network, secondly is when the update service captures a new event from the Blockchain that is triggered by the manufacturer on uploading the new firmware. Figure 2 shows what happens on each initiation process. In this phase, we focus more on session key exchange. The device starts by sending the initiation message, which consists of the identifier, which is unique for this message and the nonce value.

The nonce is a number that is used only once to avoid any replay attack on the session key exchange and for any confidential data. The update service then generates session keys and sends them as a downlink message. It is a best practise not to use the same encryption keys for a long period therefore, we do not intend to use UDTKey for encryption or decryption of any confidential data for the entire update session but instead the UDTKey is used to generate new set of keys for encryption and decryption of data. AES key and MAC key are of both 16 bytes each. We consider using symmetric based cryptography AES and MAC because, are suitable for constrained devices compare to digital signature that based on asymmetric cryptography this has been shown is several studies [16][17].

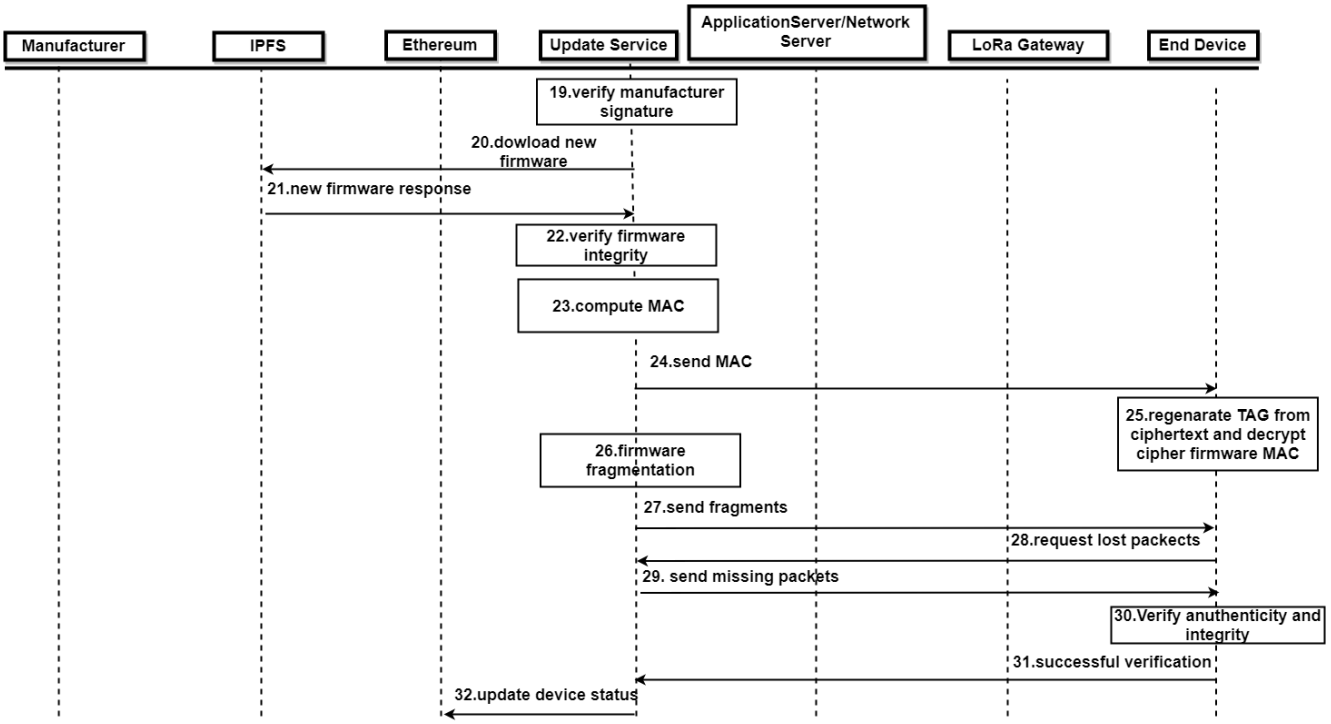

Fig 3.Cont. Interaction of system components.

Both MACKey and AESKey are shared encrypted using the UDTKey that was shared earlier with the end device. The AESKey is used to encrypt and decrypt integrity hash or tag rather any confidential message that will be shared by an update service to the end device. We intend to provide the integrity and authenticity on the end device through the MAC algorithm called Hash-based Message Authentication Code (HMAC) in SHA256 mode.

ID	IV Nonce	DevNonce	ServNonce	AESKey	MACKey
1	4	4	4	16	16

Fig 4. Session key exchange format.

The device obtains payload structured from Figure 4 and decrypts the session keys with UDTKey and most importantly; it checks the nonce values for any replay attack. The device responds with the payload showed in Figure 5 which consist of the device, service nonce and the current firmware version.

ID	IV Nonce	DevNonce	ServNonce	Version
1	4	4	4	N

Fig 5. Device version exchange.

Once the current version is received, an update service fetches a smart contract, wallet address and device model from the Blockchain. The smart contract address is used to locate the Application Binary Interface (ABI) of manufacturer on Ethereum network. The wallet address is used for verification of the manufacturer's signature. Firmware version and model of the device is provided on the Blockchain to check for any new firmware update and if the update is available, the metadata is retrieved.

The second way the firmware update can be started is through an event captured on the Blockchain network. This is illustrated in Figure 2 as the second condition on the sequence diagram. In this exchange the update service, get the metadata from the events and then exchange the message shown in Figure 4.

E. The Entire Firmware Process

Figure 3 illustrates the entire firmware update process. At this stage, the session keys are exchanged successfully and the firmware metadata is obtained successfully. The authenticity and integrity of the firmware have to be determined. The metadata contains information about where to get the actual firmware image. We have to decide or not we have been represented with the right metadata. Despite the fact that update service is connected via a secure channel with the Blockchain node, the authenticity of the metadata must be achieved. Firmware manufacturer has already signed the firmware with the private key; hence, the corresponding public key is then used to verify the authenticity of the firmware. However, because Ethereum Blockchain is not based on RSA signature but, is based on ECDSA therefore, the verification process is little different. Every entity on the Blockchain has three unique set of keys/addresses. The private key, public key and the wallet address. We perform signature verification by feeding a function with the message (metadata) together with the signature (manufacturer's signature). The function will then generate a wallet address that signed the metadata in this case, is the manufacturer's wallet address. The wallet address produced from the function is matched against the one registered by the device earlier on the Blockchain. If the addresses are the same then we trust the metadata and continue with the firmware updates.

Metadata consist of the IPFS that is used to download the firmware image. Even though the content is served over a secure channel between the IPFS node and the update service, it is a necessity to achieve integrity. The update service re-computes the SHA256 hash and compares it with the hash on the metadata. If two hashes are the same, that means we have successfully achieved the integrity and we continue with the firmware update. Since, the firmware updates are automated meaning, the update service continuously listen for any new incoming firmware updates for the devices registered to it. Therefore, it is mandatory to perform cryptographic operations on top of the application server before; the firmware is fragmented and sent to the device. This serves as an additional layer of security at the application level to provide encryption, integrity and authentication based on asymmetric cryptography.

LoRa end device needs to verify the integrity and authenticity of the firmware. For that, we use MAC algorithm based on hashing which is HMAC-SHA256. Before the firmware image is sent over LoRaWAN the exchange of MAC tag of 16 bytes takes place to determine the integrity and authenticity of the firmware. Note, the MAC or tag produced by the HMAC is of 32 bytes however, instead of sending the entire MAC we send the truncated MAC of first 16 bytes. Usually this kind of integrity and authenticity is done before the bootloader takes over. The MAC tag was encrypted using the AESKey and signed with the MACKey, which was exchanged earlier. A device receives the MAC exchange message showed in Figure 6 then validate the authenticity and the integrity using an earlier shared MAC session key.

ID	TAG	IV Nonce	DevNonce	ServNonce	MAC
1	8	4	4	4	16

Fig 6. MAC exchange.

Once the MAC is successfully exchanged, we do the fragmentation, which is based on the spreading factor (SF) or data rate used. For instance, if SF12 is used each fragment must not be more than 51 bytes. Each fragment is saved on the storage or flash memory. In case of the lost fragments the flash memory is scanned to see, any lost fragments then send the list of fragments missing. After the device has received all the fragments including the ones of packets lost, it performs the integrity and authenticity check by recomputing the firmware image MAC tag and match against the one received from the update service in Figure 6. If MAC do not match, the update process disregard firmware and abort the process. Generally, the bootloader takes control of verifying the firmware before it is flashed on the device. However, for the scope of this work we do not focus on the security of the bootloader after the firmware has been successful verified by the application.

V. Results

A. Implementation

The experiments were performed on UBUNTU 19.04 with Intel Core i5-4300 CPU @ 2.6GHz. The proposed architecture utilizes public Ethereum Blockchain network, a public Blockchain is chosen because, of the nature of firmware updates which is meant to share the firmware image and it metadata publicly. The Ethereum Blockchain smart contract drives the firmware update and ensures the security during the firmware process. Smart contact is implemented with the solidity programming language and deployed on the Rinkeby test network. The infura service node is used to give the access to the Blockchain network. A web-based manufacturer interface is implemented to interact with the Blockchain network via the infura Blockchain node. Figure 7 shows the implemented interface that allows the manufacturer to sign firmware, upload the firmware image to the decentralized IPFS network and publish metadata to the Blockchain network.

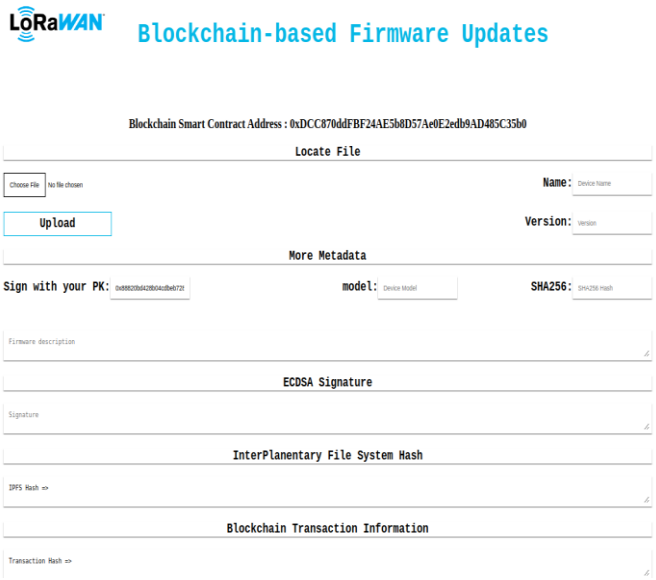

Fig 7. Manufacturer interface.

In handling LoRa packets, we run our own private LoRaWAN servers using the TTN stack v3.8.3. These servers connect to the RAK831 gateway that forwards LoRa packets to the LoPy device. The update service is implemented in python and interacts with the Blockchain network via web3 interface. The firmware image needs a place to be stored and we use IPFS infura service node that is connected to the IPFS decentralized network. The firmware mechanism is tested on the LoPy LoRa end device from Pycom, which uses EU region with the channel random selected.

TABLE I. Evaluation Paremeters.

Parameters	Values
Rx1 Window	Both downlink and uplink
Region	EU (channels duty cycle 1%)
Gateway	1
Device Antenna Gain	2 dBm
Gateway Antenna Gain	0 dBi
Bandwidth	LoRa.BW_125KHZ
Spreading Factor	7 - 12

B. Evaluation

We evaluate the performance based on the device operating in class A mode. This mode requires an uplink message for a firmware fragment to be received that therefore, is considered to be not a good mode for firmware updates, however we

examine the cost involved when device is operating on class A by applying the delta update. We also evaluate the performance of Ethereum Blockchain and execution cost of Blockchain update operations. Our smart contract consists of four main methods that are presented on Table II.

TABLE II. EXECUTION COST OF BLOCKCHAIN OPERATIONS.

Methods	Gas for execution	Transaction fee Ether
addMetadata(372B)	521,049	0.000521049
addMetadata(744B)	537,533	0.000537533
addMetadata(1.16KB)	588,670	0.00058867
registerDevice()	50,718	0.00405744
updateDevInfo()	28,968	0.00231744
getMetadata()	0	0
getDevInfo()	0	0
getDevsInfoByModel()	0	0

From Table I, we can observe that, as the metadata increases in size the gas needed to mine the metadata on the Ethereum network increases as well. For instance, when the metadata size increases from 300 bytes to 600 bytes the gas also increases from 521,049 to 537,533, which means extra 16,484 gas is required. This is even true for other transactions which including registering the device and updating the device information after the successful update. However, when the update service retrieves the data from the Blockchain there are no transaction fees. This is because there is no transaction being added on the Blockchain based on get operations.

We also study the effect of the firmware sizes, which are 1KB, 2KB, 3KB, 4KB and 5KB. Figure 8 depicts that the number of fragments tends to decrease from the small SF to high SF. This is because of the limitation on the maximum payload one is allowed to transmit on a given data rate or SF and the region the device is operating in. The maximum application payload for SF12, 11, 10 should not be more than 51 bytes. SF9 limits no more than 115 bytes and finally for SF8 and SF7 only fragments less than 222 bytes are allowed. However, 45,104 and 204 were used to ensure that the payload falls under a SF or data rate. Figure 9(a) shows fragment size used per each SF and airtime of each fragments size while Figure 8 represents the total number of fragments needed to be sent depending of the SF and the firmware size. It is observed that from Figure 9(b) for the same fragment size the airtime tends to differ for each SF. For instance, using SF12 and SF11 on the fragment size of 45 bytes the airtime is higher on SF12. We conclude that at the highest SF the firmware fragments tend to increase, which leads the higher or increase in airtime.

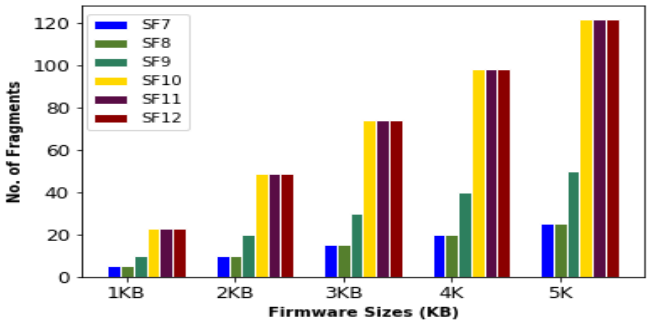

Fig 8. Number of firmware fragments.

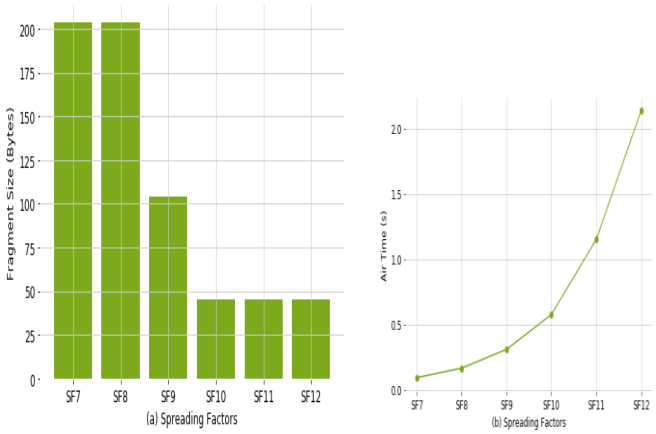

Fig 9. Airtime with corresponding fragment.

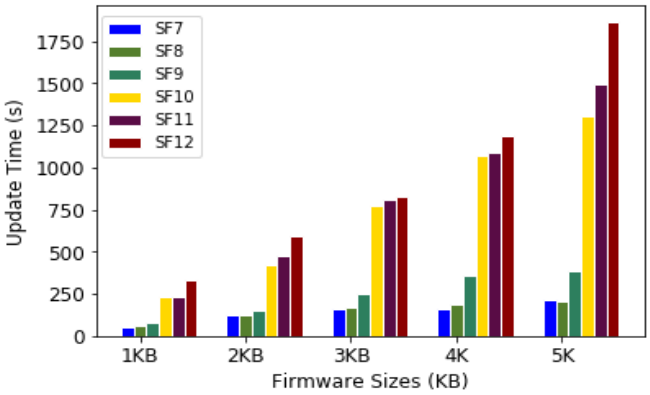

Fig 10. Time taken to update.

Firmware size has a huge impact on the update time. The update time refers to the time where the first firmware fragment was sent to the end device to where the entire firmware is verified by the end device. It is observed that from Figure 10 the update time increases as firmware size increases. For instance, applying 5 KB of firmware using SF 7 it only takes 3.55 minutes (213 seconds) whereas, using the SF 12 it takes 31.1 minutes (1866 seconds). This increase of update time is due to the airtime of each fragment for a particular SF or is due to the duty cycle limitation since, the next firmware fragment has wait for the next opportunity to be sent. We can conclude that the larger the firmware image, the more time would take to update the LoRa end device.

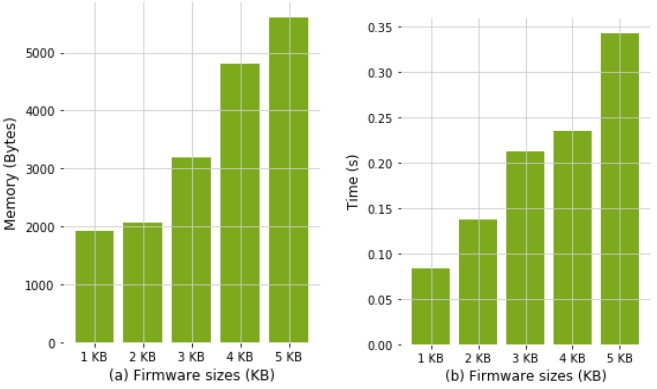

Fig 11. Memory usage on the end device and execution of cryptographic HMAC operation.

Applying firmware update on the constrained device and performing the cryptographic operation could be very expensive in resources therefore, it is important to look at the cryptographic costs involved when securing the device during the update process. The LoRa end device had 56704 KB of RAM available before the firmware update. Figure 11(a) and Figure 11(b) show both memory consumption and time take for each cryptographic operation.

The MAC verification time of the firmware image tends to increase as the image sizes increase which also consumes more memory for verifying the image. This because, the entire firmware is loaded from the flash memory/storage and read in small chunk sizes which are appended to form whole binary that serves as an input the HMAC-SHA256. Therefore, this leads the firmware size to be directly proportional to both time and memory. Since end device consist of approximately 56704 KB of RAM this means the mechanism can utilize 10.1% of RAM in verification of firmware image of 5KB.Now, considering constrained devices class-0 and class-1. Constrained devices belonging to these classes consists of RAM and flash of tens or hundreds of kilobytes. Therefore, if a constrained device consists of the RAM approximately to 10KB the MAC verification could a take half of the memory when updating the 5BK image.

VI. Conclusions And Future Work

It is inevitable that bugs and vulnerabilities are only going to be discovered after IoT devices have been deployed to the field. This means that the device manufacturers are expected to release new firmware versions to fix bugs, to improve device functionality. In this paper, we presented a Blockchain-based firmware update mechanism to enhance LoRaWAN security during the firmware update process. The orchestration of the whole process is performed by the update service implemented to serve as an extra layer of security during the firmware update.

We evaluated a proposed Blockchain-based mechanism showing the impact of updating LoRaWAN class A device. The results showed that the firmware update size or fragment has great impact in update time for a certain spreading factor used. For example, applying 5 KB of firmware using SF 7 can takes about 3.55 minutes whereas, using the SF 12, it takes 31.1 minutes; this could be even more for larger firmware size. The firmware update Blockchain smart contract operations demonstrated the cost in involved as the metadata size increases, which also leads to more gas required to store metadata on the ledger. The firmware verification measurements indicate that using HMAC-SHA256 with 128-bit key will required more RAM when the firmware image gets larger. In future, we intend to test the firmware update mechanism on large number of devices; LoRa Alliance has come up with the number of new specifications that makes firmware update process easier. These specifications include multicast, fragmentation, clock synchronisation and these can be simple incorporated with our Blockchain based update service to deliver the firmware update to large number of devices.

References

[1] K. R. Ozyilmaz and A. Yurdakul, "Designing a Blockchain-Based IoT with Ethereum, Swarm, and LoRa: The Software Solution to Create High Availability with Minimal Security Risks," *IEEE Consum. Electron. Mag.*, vol. 8, no. 2, pp. 28–34, 2019.

[2] J. Jongboom and J. Stokking, "Enabling firmware updates over LPWANs," *Embed. World Conf.*, 2018.

[3] A. Gupta, *The IoT hacker's handbook [electronic resource]: A practical guide to hacking the internet of things / Aditya Gupta*. 2019.

[4] OWASP, "OWASP Top 10 Internet of Things," *Salem Press Encycl. Sci.*, pp. 5–7, 2018.

[5] E. Aras, G. S. Ramachandran, P. Lawrence, and D. Hughes, "Exploring the security vulnerabilities of LoRa," *2017 3rd IEEE Int. Conf. Cybern. CYBCONF 2017 - Proc.*, 2017.

[6] S. (CSA) Khemissa, "Using Blockchain Technology to Secure the Internet of Things," p. 26, 2018.

[7] S. M. Danish, M. Lestas, W. Asif, H. K. Qureshi, and M. Rajarajan, "A Lightweight Blockchain Based Two Factor Authentication Mechanism for LoRaWAN Join Procedure," *2019 IEEE Int. Conf. Commun. Work. (ICC Work.*, pp. 1–6, 2019.

[8] J. Lin, Z. Shen, and C. Miao, "Using Blockchain Technology to Build Trust in Sharing LoRaWAN IoT," *Proc. 2nd Int. Conf. Crowd Sci. Eng. - ICCSE'17*, no. February, pp. 38–43, 2017.

[9] A. Durand, P. Gremaud, and J. Pasquier, "Resilient, crowd-sourced LPWAN infrastructure using blockchain," *CRYBLOCK 2018 - Proc. 1st Work. Cryptocurrencies Blockchains Distrib. Syst. Part MobiSys 2018*, pp. 25–29, 2018.

[10] J. Benet, "IPFS - Content Addressed, Versioned, P2P File System," no. Draft 3, 2014.

[11] LoRa Alliance™, "LoRaWAN Application Layer Clock Synchronization Specification v1.0.0," pp. 1–13, 2018.

[12] K. Abdelfadeel, T. Farrell, D. McDonald, and D. Pesch, "How to make Firmware Updates over LoRaWAN Possible," no. February, 2020.

[13] L. Examples, "OTA update," pp. 1–7, 2020.

[14] B. Lee and J. H. Lee, "Blockchain-based secure firmware update for embedded devices in an Internet of Things environment," *J. Supercomput.*, vol. 73, no. 3, pp. 1152–1167, 2017.

[15] S. A. Nanopoulos, "Código Network : a Decentralized Firmware Update Framework for IoT Devices," 2018.

[16] A. Report, "Secure In-Field Firmware Updates for MSP MCUs," no. November, pp. 1–13, 2015.

[17] J. King, "A Distributed Security Scheme to Secure Data Communication between Class-0 IoT Devices and the Internet," p. 58, 2015.