A Fault-Tolerant and Secure Architecture for Key Management in LoRaWAN based on Permissioned Blockchain

VICTOR RIBEIRO1, RAIRI HOLLANDA2, ALEX RAMOS3, JOEL J. P. C. RODRIGUES4,5

1University of Fortaleza (UNIFOR), Fortaleza 60811-905, Brazil (e-mail: pasknel@edu.unifor.br)
2University of Fortaleza (UNIFOR), Fortaleza 60811-905, Brazil (e-mail: raimir@unifor.br)
3Federal Institute of Ceará (IFCE), Fortaleza 60040-531, Brazil; (e-mail: alex.ramos@ifce.edu.br)
4College of Computer Science and Technology, China University of Petroleum (East China), Qingdao 266555, China (e-mail: joeljr@ieee.org)
5Instituto de Telecomunicações, 6201-001 Covilhã, Portugal

Corresponding author: Victor Ribeiro (e-mail: pasknel@edu.unifor.br).

This work was partially supported by Morphus Labs (Brazil), by FCT/MCTES through national funds and when applicable co-funded EU funds under the Project UIDB/50008/2020; and by Brazilian National Council for Scientific and Technological Development - CNPq, via Grant No. 306389/2020-7 and 313036/2020-9.

ABSTRACT Low-Power Wide-Area Network (LPWAN) is a new type of wireless technology that offers long range communication for devices in the Internet of Things (IoT) and LoRaWAN is one of the main technologies currently available to enable LPWAN environments. In the LoRaWAN architecture, the Join Server is a key component and is responsible for security tasks, such as authentication and key management. However, the Join Server acts as a Single Point of Failure (SPOF) since all encryption keys are stored centrally. Then, this paper presents a secure and fault-tolerant architecture to increase the levels of security and availability in LoRaWAN. A permissioned blockchain and smart contracts are used to replace the Join Server and solve the SPOF problem. A working prototype was created using open-source tools in order to evaluate the feasibility of the proposed architecture. Additionally, the performance of a blockchain network was analyzed in a cloud environment under multiple workloads and fault-tolerance experiments were performed to evaluate the impact of network failures. The results show a trade-off between availability and performance when choosing the number of blockchain peers in small scenarios. However, this behavior is reversed in large scenarios where the performance of multiple peers is best suited.

INDEX TERMS Blockchain, Hyperledger, Internet of Things, IoT, LoRaWAN, LPWAN, Security.

I. INTRODUCTION

INTERNET of Things (IoT) is a revolutionary paradigm which aims to provide ubiquitous connectivity between smart objects and humans. The IoT brings many benefits to society and has the potential to enable complex scenarios, such as smart cities and smart agriculture. Low-Power Wide-Area Network (LPWAN) is a new type of wireless technology that offers long range communication for IoT devices. This technology provides the ability to transmit messages across many miles and very low power consumption for resource-constrained IoT devices. However, these advantages come with trade-offs since there can be restrictions on message size, number of transmissions, and low data rates. Therefore, LPWAN solutions are ideal for scenarios in which long time delays are tolerated and IoT devices only need to send a few messages per day.

Among the available LPWAN technologies, LoRa is one of the most popular solutions used in the creation of LPWAN environments. LoRa consists of a proprietary modulation technique. LoRaWAN is an open standard that defines a MAC layer protocol which is built on top of LoRa. The LoRaWAN architecture is based on a star topology with multiple gateways and servers that process messages sent by end-devices. The Join Server is one of the main components of LoRaWAN and is responsible for performing security tasks, such as authentication and key management. Availability is a security challenge present in LPWAN environments. Some components of the LoRaWAN architecture are susceptible to attacks and may act as a Single Point of Failure (SPOF). Since all encryption keys are stored centrally, the Join Server can be considered as a potential risk to availability.

Blockchain is an innovative technology that considers a
The main contributions of this paper are described as follows:

1. Increased availability and security in LoRaWAN environments through a blockchain-based architecture.
2. In order to validate the feasibility of the proposed architecture, a working prototype was created using open-source tools and commodity hardware.
3. No updates are required to the LoRaWAN protocol and device firmwares since the necessary modifications are performed only to the LoRaWAN servers. Moreover, the proposed architecture can be applied to the latest version of LoRaWAN.

To handle these issues, this paper proposes a secure and fault-tolerant architecture to increase the levels of security and availability in LoRaWAN. The Join Server is replaced by a smart contract in a permissioned blockchain, therefore, solving the SPOF problem due to the decentralized design of the blockchain. In the proposed architecture, the smart contract implements key management tasks and authentication procedures performed previously by the Join Server. The main contributions of this paper are described as follows:

II. RELATED WORK

Many works can be found in the literature proposing the integration between blockchain and LPWAN.

Authors of [4] propose an infrastructure in which LoRa gateways act as clients to a blockchain network. The proposed solution is implemented using real LoRa devices and the Ethereum blockchain. However, the authors do not describe the security features of the proposed architecture. Additionally, there are no details about the integration between the Application Server and the blockchain nodes. A similar proposal, but designed for a pollution monitoring application can be found in [5].

In [6], [7], the authors propose the application of blockchain capabilities to LoRaWAN deployments. In the proposed infrastructure, multiple Network Servers form a decentralized network and perform blockchain tasks, such as hashing, transaction verification, and block chaining. The proposed solution is able to check the validity of transactions performed in the network. However, the authors do not discuss the security aspects of the architecture and no implementation of the proposed solution was presented.

A blockchain-based access control system for IoT is proposed in [8]. The proposed solution uses three smart contracts: Device Contract (DC), Policy Contract (PC), and Access Contract (AC). DC is responsible for storing the URL of resource data generated by IoT devices. AC manages Associated Based Access Control (ABAC) models for admin users. AC provides the core access control functions for regular users. The three smart contracts combined enable a distributed and dynamic access control system. The authors implemented a permissioned blockchain with Hyperledger Fabric. Moreover, the authors performed simulations under multiple workloads in order to evaluate the proposed system.

A two-factor authentication mechanism based on blockchain is proposed in [9]. The proposed solution combines the standard LoRaWAN authentication (Join procedure) with a blockchain-based authentication. An agent node mediates the communication between the LoRaWAN nodes and the blockchain network. The agent node also executes a smart contract which is responsible for the second authentication step. The authors implemented the proposed scheme with the Ethereum blockchain. In addition, a performance analysis was performed in order to evaluate the proposed solution with respect to latency and throughput. Results show that a considerable amount of delay is added during the first Join procedure, due to common mining tasks that occur in the Ethereum blockchain. However, the Join Server is still present in the LoRaWAN network and firmware updates are required for end-devices in order for the proposed solution to work.

In [10], the authors propose a blockchain-based architecture to increase availability and security in the join procedure at LoRaWAN. The key management is shared between Join Server and Network Server, which are all nodes in a permissioned blockchain. The authors validate their proposal by implementing a prototype using ChirpStack and Hyperledger, then evaluating its performance and security.

To protect the join procedure against jamming and replay attacks, a blockchain-based LoRaWAN architecture is also proposed by the authors in [11]. In the proposed architecture, the blockchain nodes are the the Network Servers in the LoRAWAN. The Join Server handles join requests by using smart contracts to read/write authentication data in the blockchain network. This approach allows to build a secure key management solution for LoRAWAN. The authors implement their solution using Ethereum blockchain and perform simulations to validate it. The obtained results demonstrate the efficiency of the proposed system under a load of 30 join requests generated from 1000 simultaneous end-devices.

In our previous work [12], the Join Server was integrated with a permissioned blockchain to ensure high availability.
and secure data storage. In contrast to our previous paper, in the present work, the Join Server has been fully replaced by a smart contract which is executed by multiple peers deployed in the blockchain network. Furthermore, this work deepens the evaluation of the proposed architecture by conducting performance and fault-tolerance tests in a cloud environment. Finally, the implemented prototype is compatible with the latest LoRaWAN specification.

III. LOW POWER WIDE AREA NETWORKS

This section provides an overview of the LPWAN paradigm. First, the characteristics of LPWAN environments are explained. Then, the architecture and features of LoRa networks are presented. Finally, the security features of LoRaWAN are described.

A. LPWAN OVERVIEW

LPWANs are emerging technologies that offer connectivity to multiple devices deployed in large environments (e.g., urban and rural areas). LPWAN technologies are ideal for scenarios with low cost devices that must transmit small packets over long distances with no time constraints. Devices can be deployed in remote and harsh environments. Among the currently available LPWAN solutions, we can mention LoRa, Sigfox, Weightless, and NB-IoT as some of the main technologies in this area [13], [14]. LPWAN solutions can be divided based on the frequency bands used: unlicensed (e.g., Sigfox and LoRa) and licensed (e.g., NB-IoT).

There is a clear trade-off between communication range and data rate for LPWAN technologies. End-devices can transmit messages over great distances at the expense of very low data rates and high latency [15]. LPWANs are built based on star topology architecture. End-devices are directly connected to one gateway using single-hop communication. Even though LPWAN solutions offer bidirectional communication between end-devices and network servers, most of the communication is uplink oriented.

LPWAN solutions offer more advantages, in terms of cost and battery lifetime, when compared with mesh networks and traditional cellular networks. As opposed to mesh networks, end-devices can sleep most of the time in LPWANs, since packets don’t need to be routed between devices, therefore, power consumption is greatly reduced. The use of star topology can also facilitate the maintenance and deployment of IoT networks, and simplify the design of end-devices (as most of the complexity will be performed by network devices, such as gateways and network servers).

B. LORA

LoRa is one of the main technologies currently available to enable LPWAN scenarios [16]. LoRa can be divided in two parts: physical layer and MAC layer. The physical layer is composed of a proprietary specification, developed by Semtech, based on Chirp Spread Spectrum (CSS) [17]. LoRa can be used with multiple unlicensed bands, such as 433MHz, 868MHz, and 915MHz. The MAC layer, known as LoRaWAN, is an open specification maintained the LoRa Alliance and is currently at version 1.1 [18].

The network architecture is based on a star-of-stars topology, as illustrated in Figure 1. There are five types of nodes: end-devices, gateways, Network Server (NS), Application Server (AS), and Join Server (JS). End-devices collect information from the local environment and communicate directly with one or multiples gateways using LoRa physical layer. Each gateway receives new uplink messages and forwards them to NS through a traditional IP network. NS is responsible for multiple tasks, such as packet routing, roaming, and frame authentication. AS receives all uplink messages and provides high-level services to the end-users. JS is responsible for carrying out the main security tasks in LoRaWAN: authentication procedures and key management.

LoRaWAN defines three classes for end-devices:

- **Class A**: Intended for devices with power restrictions (battery-operated). Offers bi-directional communication in which an uplink transmission is followed by two downlink receive windows (RX1 and RX2). The uplink window is scheduled based on the end-device needs (similar to ALOHA protocols). Downlink messages can be sent only after uplinks transmissions, therefore, servers may need to wait until the next uplink slot. Every LoRaWAN device must support Class A by default.
- **Class B**: Designed for devices which may need additional downlink windows. Gateways transmit beacon messages to synchronize with Class B devices and schedule additional receive slots.
- **Class C**: Planned for devices which may need extended receive windows and have constant power available.

C. LORAWAN SECURITY

LoRaWAN provides security features for data confidentiality, message integrity, and end-device authentication. LoRaWAN security is based on symmetric encryption and the Advanced Encryption Standard (AES) algorithm. Distinct 128-bit AES keys are used for each layer, one root key for the network layer (NwkKey), and another one for the application layer (AppKey). The root keys must be unique for each device and are later used to derive session keys. If the encryption keys of one device are compromised, the overall security of the network will not be affected. Data confidentiality is guaranteed by encrypting application data with AES.
message contains a Message Integrity Code (MIC) which is used to authenticate the sender and verify the integrity of the message.

An end-device must first authenticate in order to join a LoRaWAN network. LoRaWAN provides two authentication methods:

- **Over-the-Air Activation (OTAA):** security handshake performed between an end-device and JS. Join-Request and Join-Accept messages are exchanged during the procedure and used to derive new session keys. Each end-device must store the following data: JoinEUI, DevEUI and the root encryption keys. JoinEUI is a unique 64-bits global identifier used to inform which JS will handle the Join procedure. DevEUI is a global EUI-64 ID used to uniquely identify an end-device. During the Join procedure, new session keys are derived based on the two root keys. NwkKey is used to derive a triplet of network session keys (NwkSEncKey, FNwkSLtKey and SNwkSLtKey), while AppKey is used to derive one application session key (AppSKey). Once OTAA is completed, the new session keys will be used to secure all communications between the end-device and the LoRaWAN servers. The LoRaWAN Backend Interface specification [19] advises the use of Key Encryption Keys (KEK) and a wrapping algorithm for the secure transportation of session keys between LoRaWAN servers (e.g., JS and AS).

- **Activation By Personalization (ABP):** end-devices are pre-configured with all necessary encryption keys before being deployed to the network. Therefore, there is no interaction between JS and end-devices in ABP mode. This method offers less security since the encryption keys will remain constant (hardcoded) during the lifetime of an end-device. ABP devices are more sensitive to physical attacks, for example, if the session keys are compromised, an attacker will be able to decrypt all future communications [20]. Network administrators will have to manually update the session keys stored in each compromised device.

The Rejoin procedure was introduced in LoRaWAN 1.1 and it allows JS and NS to create a new security context for an end-device (e.g., reset session keys). The Rejoin procedure contains 3 types of Rejoin-Request messages (0, 1, and 2) and is always initiated by an end-device. Interaction with JS is restricted to Rejoin-Request messages of type 1 (the remaining types are sent only to NS).

Despite the security mechanisms previously described, LoRaWAN environments are still susceptible to some attacks. Since all root keys are stored centrally, JS acts as a Single Point of Failure (SPOF) and can be considered as a serious risk to availability. For example, if JS is compromised or becomes unavailable (e.g., Denial-of-Service attack), end-devices will not be able to join the network and send messages. Because of the decentralized design of blockchain and the opportunity to use smart contracts, the integration between blockchain technology and LoRaWAN shows great potential to handle the SPOF risk imposed by JS.

**IV. PERMISSIONED BLOCKCHAINS**

This section provides an introduction to permissioned blockchains. First, the basic characteristics of blockchain technology are explained. Second, an overview of the Hyperledger Fabric project is presented. Finally, the security features used by Hyperledger Fabric to ensure data confidentiality are detailed at the end of this section.

**A. BLOCKCHAIN OVERVIEW**

Blockchain technologies implement a distributed ledger on top of a peer-to-peer network infrastructure. Blockchain eliminates the need for a central authority and third parties due to its distributed design. Cryptography primitives are used by blockchain technologies to ensure the immutability of data added to the distributed ledger. Moreover, each transaction sent to the blockchain is validated by the participating peers using a consensus algorithm. This process ensures the auditability of all records in the ledger.

Blockchain technologies can be divided in two categories: public and permissioned. In public blockchains, any node in the world is able to participate in the consensus process and every transaction can be seen by all peers. Bitcoin and Ethereum are examples of public blockchains. In permissioned blockchains, the consensus process is restricted to authenticated peers. Therefore, transactions can be seen only by a group of trusted peers. Hyperledger Fabric is an example of permissioned blockchain.

When compared with classic approaches (e.g., load-balancing and redundancy [21]–[23]), a permissioned blockchain is also able to provide high-availability since all network peers replicate the same information [24]. Therefore, if one peer becomes unavailable, data can still be found in the remaining peers. For example, performance evaluations presented in [25], [26] demonstrate that Hyperledger Fabric is able to provide low response time and high availability (98%). Furthermore, a permissioned blockchain eliminates the SPOF problem, while also ensuring additional important features, such as auditability, immutability, and non-repudiation.

Hyperledger Fabric has been chosen for the proposed architecture due to the following reasons:

- **Performance:** The use of lightweight consensus algorithms results in better performance when compared to other blockchains [27], [28].

- **Confidentiality:** In contrast to public blockchains, a permissioned blockchain offers greater levels of data confidentiality by restricting access to a group of trusted peers.

**B. HYPERLEDGER FABRIC**

Hyperledger Fabric is a open source implementation of a permissioned blockchain. The project was initiated by IBM and is currently maintained by the Linux Foundation. The
modular architecture of Hyperledger Fabric allows pluggable components, such as membership services, privacy, and consensus algorithms. Transactions are carried out in channels which are separate ledgers where access is restricted to authorized peers. Hyperledger Fabric supports the execution of smart contracts (named “chaincode”) which are distributed applications that can read and write data from the blockchain, and enforce agreements. Chaincodes can be implemented with general purpose languages, such as Go, Java, and Javascript. Data is stored and retrieved from the blockchain as key-value pairs. In comparison to public blockchains, Hyperledger Fabric features no transaction costs since it does not rely on a cryptocurrency.

Nodes are divided in the following roles:

- **Client**: an application that creates and sends transaction proposals to endorsing peers.
- **Peer**: this role can be divided in two types: endorsers and committers. Endorsing peers validate, execute, and endorse transaction proposals from clients (only endorsing peers have chaincodes installed). Committing peers receive new blocks from the ordering service, verify if a transaction is correctly endorsed, and commit the blocks to the ledger.
- **Ordering Service**: Ordering nodes receive endorsed transactions from client applications, order new transactions into blocks, and send them to committing peers. The main objective of the ordering service is to achieve consensus in the network. Hyperledger Fabric offers 3 implementations for the ordering service: Solo, Kafka, and Raft. Solo runs in a single node and is used for testing purposes. Kafka is a distributed messaging system which follows the publish-subscribe pattern [29]. Raft is a consensus algorithm for managing replicated logs based on leader elections [30].
- **Membership Service Provider (MSP)**: defines rules for how the identity of each network component is validated. The MSP uses a Certificate Authority (CA) to issue digital certificates for each node.

The Execute-Order-Validate model dictates how the transaction flow is executed in Hyperledger Fabric (presented in Figure 2). The main steps of the Execute-Order-Validate model are described as follows:

1. A client application sends a transaction proposal to one or more endorsing peers (the minimum required number of endorsing peers is defined by an endorsement policy). The transaction is created to invoke a chaincode function and perform a set of operations to read or update values from the ledger.
2. The endorsing peers verify if the client is authorized to perform operations in the channel and check if the transaction is valid. The transaction proposal is executed by the chaincode installed in the endorsing peers. Finally, a signed transaction proposal is returned to the client with a response value and read/write sets (the global state of the blockchain is not updated at this point).
3. The ordering service receives a signed transaction proposal from the client and creates a new block of transactions based on a consensus protocol.
4. The new block is delivered to the committing peers during the final step. At this point, two methods are used to validate each transaction: Validation System Chaincode (VSCC) and Multiversion Concurrency Control (MVCC). VSCC validates the endorsement signatures present in the transaction against the endorsement policy defined for the invoked chaincode. MVCC checks if the versions of the keys read by a transaction are the same as the versions found in the current global state. Once MVCC and VSCC are confirmed, the new block is added to the blockchain.

**C. PRIVATE DATA COLLECTION**

Channels are used by Hyperledger Fabric to share data between two or more groups. However, there may be some scenarios in which confidentiality is a mandatory requirement and sensitive data must be restricted to only one group or a specific subset. For example, patient data must be accessed
only by authorized parties in smart healthcare. Private Data Collection (PDC) is a security feature implemented by Hyperledger Fabric which allows a subset of groups to commit and query sensitive data in a secure manner. The use of PDCs allows data to be accessed only by authorized peers and avoids the creation of additional channels.

The steps performed in the Execute-Order-Validate model are slightly modified if a PDC is being used by Hyperledger Fabric. Figure 3 presents an example of the modified transaction flow in a scenario with two groups (GRP1 and GRP2). A channel is shared between the two groups for public data and a PDC is used to ensure the confidentiality of sensitive data that should be accessed only by members of GRP1. Each authorized peer has a private state database to store sensitive data. The chaincode installed in authorized peers is able to access data from the private state database.

The main steps of the transaction flow with private data are described as follows:

1) A transaction proposal is sent to an endorsing peer of an authorized group. Private data is sent in the transient field of the transaction. The transient field is used in situations where the user needs to pass sensitive data to the chaincode without keeping it in the transaction record.

2) The transaction proposal is validated by the endorsing peer and simulated by the installed chaincode. Private data is stored by the endorsing peer in a temporary local storage and shared with the remaining authorized peers via gossip protocol. The endorsing peer returns a signed transaction proposal (with the hash of the private data) to the client.

3) The client sends the signed transaction proposal to the ordering service. Since the transaction proposal contains only the hash of the private data, no sensitive information is sent to the ordering service. The transaction is added to a new block by the ordering service and sent to all peers.

4) At the commit phase, the authorized peers will validate the private data (stored in the temporary local storage) with the hashes found in the public block. Once the validation is complete, sensitive data is moved from the temporary storage to the private state database.

V. PROPOSED ARCHITECTURE

In this section, we present the proposed architecture for LoRaWAN. First, the network topology and chaincode functions are described. Then, the message workflow of the proposed architecture is detailed. Finally, the components used in the creation of a working prototype are listed.

A. NETWORK TOPOLOGY

Figure 4 presents the proposed network architecture. JS has been replaced by a chaincode which is responsible for performing security and key management tasks, such as handling the OTAA procedure and managing the root encryption keys. The chaincode and NS are divided into two different groups: GRP1 and GRP2. The chaincode is installed in multiple endorsing peers from GRP1 in order to ensure availability. NS acts as a client application from GRP2 and is able to interact with the chaincode. The administrator of GRP1 is responsible for initiating the chaincode and registering the encryption keys from new devices. During registration activities or updates, the root keys are always passed in the transient field of a transaction and stored in the PDC of GRP1 (which prevents NS from having direct access to the encryption keys).

A Public Key Infrastructure (PKI) is used by Hyperledger Fabric to issue digital certificates and verify the identity of each node. Confidentiality of communications between the trusted peers is guaranteed by the use of Transport Layer 6.
Security (TLS).

B. CHAINCODE DESCRIPTION

The chaincode is responsible for performing tasks similar to a traditional JS, such as key management and authentication procedures. Moreover, the chaincode implements a set of access control rules to verify if an user is authorized to access certain functions. Listing 1 presents the `DeviceKeys` struct used by the chaincode to store and handle the root encryption keys.

```
Listing 1. DeviceKeys data structure

type DeviceKeys struct {
    CreatedAt   time.Time
    UpdatedAt   time.Time
    DevEUI      [8]byte
    NwkKey      [16]byte
    AppKey      [16]byte
    JoinNonce   int
}
```

The `DeviceKeys` struct contains the following values:

- **Timestamps**: Tracks time of creation and updates of `DeviceKeys` objects.
- **DevEUI**: Unique device identifier (8-byte array).
- **NwkKey**: Network root key (16-byte array).
- **AppKey**: Application root key (16-byte array).
- **Join-Nonce**: Counter value used during OTAA procedure to create Join-Accept messages.

The chaincode implements the following functions:

- **InitChaincode**: This function is restricted to the administrator of GRP1 and it should be invoked to initialize the chaincode with a Key Encryption Key (KEK), which must be pre-shared with AS and will be used later to encrypt AppSKey. The chaincode expects KEK to be passed in the transient field of the transaction and it will be stored in the PDC of GRP1.
  - **CreateDeviceKeysPrivate**: Stores a new pair of encryption keys in the PDC. This function is restricted to the administrator of GRP1 and it should be invoked once a new end-device joins the LoRaWAN network. In order to call this function, a new `DeviceKeys` object must be passed in the transient field of a transaction.
  - **GetDeviceKeysPrivate**: Receives a `DevEUI` value as an input parameter and returns the corresponding `DeviceKeys` object from the PDC. This function is restricted to the administrator of GRP1.
  - **UpdateDeviceKeysPrivate**: Updates a `DeviceKeys` object from the PDC. This function is restricted to the administrator of GRP1 and it should be invoked whenever the root encryption keys of an end-device needs to be changed. In order to call this function, a `DeviceKeys` object must be passed in the transient field of a transaction.
  - **RevokeDeviceKeysPrivate**: Receives a `DevEUI` value as an input parameter and removes the `DeviceKeys` object from the PDC. This function is restricted to the administrator of GRP1 and it should be invoked whenever an end-device leaves the LoRaWAN network (e.g., decommissioned devices).
  - **HandleJoinRequest**: Receives a serialized Join-Request message as input parameter, derives a new set of session keys, encrypts AppSKey with the pre-shared KEK, and returns the contents of the JoinAns message. This function must be invoked by NS during an OTAA procedure.
C. WORKFLOW

The overall workflow of the proposed architecture can be divided in 4 steps: chaincode initialization, end-device registration, OTAA procedure, and end-device decommissioning.

**Chaincode initialization:** the InitChaincode function is invoked by the administrator of GRP1 to initialize the chaincode with a pre-shared KEK which will be used to ensure the secure delivery of AppSKey to AS.

**End-device registration:** prior to the deployment of a new end-device, its root keys must be properly registered. First, the administrator of GRP1 must invoke the RegisterDeviceKeys function from the chaincode. In order to avoid the exposure of sensitive data, the root keys are always passed in the transient field of the transaction. Then, the new root keys are stored in the PDC of GRP1.

**OTAA procedure:** NS must communicate with the chaincode in order to derive a new set of session keys and generate a valid JoinAns message. Figure 5 presents the message flow that occurs during the OTAA procedure in the proposed LoRaWAN architecture. The following steps describe the process performed for OTAA:

1) In the fist step, the end-device initiates the OTAA procedure by sending a Join-Request message to NS. This message is composed of three fields: JoinEUI, DevEUI, and DevNonce. The DevEUI field is an unique identifier value for the communicating end-device. The DevNonce field is a 2-bytes counter value which will be used during the creation of the session keys in a posterior moment.

2) The Join-Request message is received and validated by NS. Then, NS invokes the HandleJoinRequest function from the chaincode (detailed in Algorithm 1).

3) At this point, the chaincode retrieves the DeviceKeys object (associated to the DevEUI value) and the pre-shared KEK from the PDC of GRP1. Once the result is returned, the chaincode derives a new set of session keys, and encrypts AppSKey with the pre-shared KEK. Then, a JoinAns message, containing the derived session keys, is transmitted to NS.

4) NS returns a Join-Accept message (extracted from the payload of the JoinAns message) to the end-device.

5) The end-device receives a Join-Accept message from NS and derives the new session keys. At this moment, the end-device is ready to send data packets to AS.

6) Once the first data packet is received by NS, the encrypted AppSKey is delivered to AS and the pre-shared KEK is used to decrypt AppSKey.

7) The data packet is then forwarded from NS to AS. Finally, AS decrypts the payload with AppSKey.

**End-device decommissioning:** the RevokeDeviceKeysPrivate function should be invoked whenever a end-device is removed from the network. Even though the DeviceKeys object is removed from the PDC, the hash of the private data (which is stored in the ledger) can still be used to prove that the information existed.

---

### Algorithm 1: HandleJoinRequest

**Input:** serialized Join-Request message  
**Output:** serialized JoinAns message

1. \( \text{joinReq} \leftarrow \text{Unmarshal(joinReqBytes)}; \)
2. \( \text{devEUI} \leftarrow \text{joinReq.DevEUI}; \)
3. \( \text{dk} \leftarrow \text{GetPrivateData(devEUI)}; \)
4. \( \text{joinAns} \leftarrow \text{DeriveKeys(dk, joinReq)}; \)
5. \( \text{KeyWrap(joinAns)}; \)
6. \( \text{dk.JoinNonce} \leftarrow \text{dk.JoinNonce} + 1; \)
7. \( \text{PutPrivateData(devEUI, dk)}; \)
8. \( \text{joinAnsBytes} \leftarrow \text{Marshal(joinAns)}; \)
9. \( \text{return} \text{joinAnsBytes} \)

---

D. PROTOTYPE DETAILS

A working prototype was created as a Proof-of-Concept (PoC) using only open-source tools and commodity hardware. LoPy boards were used as class A end-devices and 1-channel LoRaWAN gateways. Each LoPy board contains an Espressif ESP32 chipset and a Semtech LoRa transceiver SX1276. The LoRaWAN servers were created using the open-source implementations from the ChirpStack project [31].

The permissioned blockchain environment was created with Hyperledger Fabric (v2.0). Golang was chosen as the programming language to implement the chaincode due to its high performance when compared to other languages [32]. Docker images are provided by Hyperledger Fabric to deploy different types of nodes (e.g., peers and orderers). The NS implementation from ChirpStack was modified in order to invoke the chaincode deployed in the permissioned blockchain.

VI. RESULTS ANALYSIS AND DISCUSSION

The performance of the working prototype was evaluated in a cloud environment. This section presents the results obtained from performance and fault-tolerance experiments under different scenarios. Additionally, this section concludes with an informal security analysis of the proposed architecture.

A. PERFORMANCE ANALYSIS

Table 1 summarizes the main settings used during the performance analysis. A Golang script was implemented to evaluate the performance of the Hyperledger environment by sending simultaneous requests to the chaincode. Each function is called multiple times and a set of metrics is collected by the performance script. The chaincode was evaluated under
different workloads: 10, 50, 100, 250, 500, 750, 1000, 2500, 5000 and 7500 simultaneous requests (for each function). In addition, the performance of the Hyperledger network was evaluated with different numbers of endorsing peers: 2, 4, and 8. The goal was to assess how the number of endorsing peers would affect performance results.

The following metrics were chosen to evaluate the performance of the proposed architecture:

- **Latency**: For each function call, latency is defined as the amount of time taken by the chaincode to process one request.
- **Throughput**: Defined as the number of requests divided by the total amount of time taken by the chaincode to process all requests in a set.

The working prototype (described in section V-D) was deployed in an Amazon Virtual Private Cloud (Amazon VPC). Multiple EC2 virtual machines were created to host the Hyperledger Fabric containers (each container was hosted on a separate server) and the LoRaWAN servers. Each virtual machine was configured as a t2.medium EC2 instance (2 vCPUs and 4 GB of RAM) running Ubuntu 18.04.

Figure 6 shows the metric values for the CreateDeviceKeysPrivate function. For up to 2500 requests, the Hyperledger network with 2 endorsing peers has lower latency.
values than the other two scenarios. However, a transition begins to occur around 3000 requests. From this point on, a smaller number of endorsing peers starts to suffer impact on its performance while processing a large number of simultaneous requests. After 3000 requests, it is possible to see that scenarios with a larger number of endorsing peers start to show better performance. A second transition occurs around 5000 requests. At this point, Hyperledger networks with 8 endorsing peers start to show better performance than the remaining scenarios. This behavior can be clearly seen through the throughput values (figure 6b) as the scenario with 8 endorsing peers shows better values than the other scenarios after 5000 requests.

Figure 7 shows the metric values for the GetDeviceKeysPrivate function. Since this function performs only read operations, it shows a different behavior when compared to the other functions. Endorsing peers don’t need to exchange messages with each other when performing read-only operations and no new blocks are added to the blockchain, therefore, the chaincode execution is much faster. For up to 1000 requests, all 3 scenarios show similar latency values. However, the values change as the number of requests increases. After 1500 requests, it can be seen that the Hyperledger network with 2 endorsing peers presents a greater degradation when compared to the other two scenarios. The scenario with 8 endorsing peers shows better values for all the metrics evaluated. Based on the values seen, we may conclude that a large number of endorsing peers will always have a better performance for read-only transactions because each request can be processed separately.

The remaining chaincode functions show a similar behavior to the CreateDeviceKeysPrivate function (the performance values are presented in figures 8, 9 and 10). As seen in figures 8a, 9a, and 10a, a smaller number of endorsing
peers present lower latency values for up to 2500 requests. However, a transition occurs as the number of simultaneous requests increases. For a large number of requests, the scenarios with 4 and 8 endorsing peers show better performance.

We conclude that there is a trade-off between performance and availability when choosing the number of endorsing peers in scenarios with a small number of simultaneous requests. For this type of scenario, a reduced number of endorsing peers shows a better performance due to less messages being exchanged between the peers. However, a small number of peers can be detrimental to data availability. A large set of endorsing peers offers a greater degree of availability at the expense of longer execution times. This situation is reversed in scenarios with a large number of requests in which the performance of a small set of peers degrades. Therefore, using multiple endorsing peers is best suited for realistic LPWAN scenarios (which are composed of a large number of end-devices and simultaneous transactions).

**B. FAULT-TOLERANCE TEST**

The goal of the fault-tolerance test is to simulate Denial-of-Service attacks and evaluate the impact in the Hyperledger environment. The cloud infrastructure used in this test is similar to the one described in section VI-A, however, a few changes were made to the environment (illustrated in figure 11).

A new virtual machine was added in order to generate failure scenarios within the working prototype. Pumba\(^1\) is a chaos testing tool that is able to create network failures in Docker containers. Pumba was used in order to kill random Docker containers running endorsing peers.

A Golang script was developed to simulate 10 simultaneous clients (each client sends continuous requests for

\(^1\)https://github.com/alexei-led/pumba

the chaincode as fast as possible). The Golang script also monitors the number of requests sent by the clients to the endorsing peers during 10 second intervals. Therefore, it is possible to analyze how the Hyperledger network behaves during the entire course of the fault-tolerance test (before and after Pumba is activated).

The fault-tolerance test consists of running the Golang script for 10 minutes in two different scenarios: 8 and 16 endorsing peers. During each experiment, two endorsing peers were killed at 300\(^{th}\) second. Each scenario was executed 15 times.

Figure 12 shows that once two endorsing peers are killed, the Hyperledger network is momentarily impacted. After a short period of time, the remaining endorsing peers are able to resynchronize and handle new incoming requests. Additionally, an increase in the number of requests processed per second can be seen in the network after Pumba is activated. This behavior can be understood based on the conclusions taken in the performance analysis section (as a smaller number of endorsing peers tends to result in higher throughput values).
C. COMPARISON WITH TRADITIONAL SETUP

The performance results obtained from the working prototype were compared with a traditional ChirpStack setup. The Golang script used during this test sent the same number of simultaneous requests as detailed in Table 1. The performance test with the traditional ChirpStack setup was executed a total of 15 times.

A comparison for the throughput values between the working prototype and the original ChirpStack setup is presented in Figure 13. Initial results show that the traditional ChirpStack setup has greater throughput values for a small number of simultaneous requests (up to 250 requests). However, the performance of the ChirpStack setup starts to degrade after 500 requests. This situation occurs because the JS in the ChirpStack setup fails to process a large number of simultaneous Join-Request messages and starts to drop OTAA requests. A performance transition happens around 3000 requests. At this point, the working prototype shows better throughput values than the traditional ChirpStack setup (for any number of endorsing peers).

D. SECURITY ANALYSIS

A secure architecture is characterized by a set of essential requirements, for example, confidentiality, integrity, and availability. One of the objectives of the proposed architecture is to provide a secure and permissioned blockchain environment which executes a smart contract. This smart contract implements the functionalities performed by a traditional JS, such as OTAA procedure and key management. The security of the remaining LoRaWAN components (e.g., gateways and AS) is out of scope of this research but a formal security analysis can be found in [33]. The following items describe how security requirements are ensured by the proposed solution:

- **Availability:** Services and data must be constantly available for clients. Availability is guaranteed in blockchain environments by storing copies of the ledger in several peers. Hyperledger Fabric allows chaincodes to be installed in multiple endorsing peers [34]. Even if a peer becomes unavailable, clients can still invoke the chaincode via the remaining endorsing peers. Therefore, the proposed architecture guarantees a high level of availability for clients and is also resilient to DoS attacks.

- **Confidentiality:** Only authorized peers must have access to sensitive data. Hyperledger Fabric restricts...
ledger access to a group of trusted peers. All communications between components (e.g., clients, orderers, and peers) is secured by the use of encryption (TLS protocol). Additionally, the proposed architecture uses PDC to ensure that the root encryption keys can only be accessed by peers from one specific group. The use of PDC works as an additional layer of security to prevent information leakage between different groups of peers. For example, even if NS is compromised, an attacker would not be able to obtain the root encryption keys from the ledger.

- **Integrity:** Data should not be modified by unauthorized parties in a secure environment. The integrity of every transaction stored in the permissioned blockchain can be verified in a secure fashion through the use of hashing mechanisms. Any attempt of data tampering by a malicious user would result in invalid hash values.

- **Authentication and authorization:** The proposed architecture ensures mutual authentication between nodes through the use of digital certificates. These certificates are signed by a CA and used to identify peers in a Hyperledger Fabric environment. Therefore, access is allowed only to trusted peers in a permissioned blockchain. The proposed architecture also implements access control rules at the chaincode level. The chaincode verifies the identity of an user through its certificate and checks if the user is authorized to access a specific function. For example, the CreateDeviceKeysPrivate function can only be accessed by one group.

- **Non-repudiation:** The proposed architecture ensures non-repudiation through digital signatures. Since all transactions are signed with a private key, the user’s public key can be used to verify the authenticity of a transaction.

**VII. CONCLUSION AND FUTURE WORKS**

This paper presents a secure and fault-tolerant architecture to improve key management in LoRaWAN networks. The proposed architecture solves the single point of failure risk posed by the JS by using a permissioned blockchain and replacing the JS with a smart contract. A working prototype was implemented using open-source tools to validate the proposed architecture. A permissioned blockchain network was created with Hyperledger Fabric and the NS implementation from the ChirpStack project was modified in order to communicate with the smart contract.

Different types of experiments were executed to evaluate the proposed architecture. First, a performance analysis was performed in a cloud environment with various numbers of endorsing peers and multiple workloads. Second, a fault-tolerance test was executed to evaluate the impact of network failures in the Hyperledger environment. The results show that there is a trade-off between performance and availability when choosing the number of endorsing peers in small scenarios. A reduced set of endorsing peers demonstrates better throughput values in situations with a few simultaneous requests (up to 2500). However, this behavior is reversed in scenarios with a large number of transactions, as the performance of larger sets of endorsing peers stands out. Therefore, we conclude that using multiple endorsing peers is best suited for realistic LPWAN scenarios.

As future works, new experiments can be performed with different implementations of ordering services and varying numbers of ordering nodes to evaluate how these points can impact the performance of the Hyperledger environment. Additionally, future research can try to integrate the AS with the Hyperledger network by using multiple channels.

**REFERENCES**

[1] Imran Makhdoom, Mehran Abolhasan, Haider Abbas, and Wei Ni. Blockchain’s adoption in iot: The challenges, and a way forward. Journal of Network and Computer Applications, 125:251–279, 2019.

[2] Bhahendu K Mohanta, Debasis Jena, Soumyashree S Panda, and Srichandan Sobhanayak. Blockchain technology: A survey on applications and security privacy challenges. Internet of Things, page 100107, 2019.

[3] Sana Moin, Ahmad Karim, Zanab Safdar, Kalsoom Safdar, Ejar Ahmed, and Muhammad Imran. Securing iots in distributed blockchain: Analysis, requirements and open issues. Future Generation Computer Systems, 109:325–343, 2019.

[4] K. R. Özylmaz and A. Yurdakul. Work-in-progress: integrating low-power iot devices to a blockchain-based infrastructure. In 2017 International Conference on Embedded Software (EMSOFT), pages 1–2, Oct 2017.

[5] S. R. Niya, S. S. Jha, T. Bocek, and B. Stiller. Design and implementation of an automated and decentralized pollution monitoring system using blockchains, smart contracts, and lorawan. In NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium, pages 1–4, April 2018.

[6] Jun Lin, Zhiqi Shen, Chunyan Miao, and Siyuan Liu. Using blockchain to build trusted lorawan sharing server. International Journal of Crowd Science, 1(3):270–280, 2017.

[7] Jun Lin, Zhiqi Shen, and Chunyan Miao. Using blockchain technology to build trust in sharing lorawan iot. In Proceedings of the 2nd International Conference on Crowd Science and Engineering, ICCSE 17, pages 38–43, New York, NY, USA, 2017. ACM.

[8] Han Liu, Dezhi Han, and Dun Li. Fabric-iot: A blockchain-based access control system in iot. IEEE Access, 8:18207–18218, 2020.

[9] 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. In 2019 IEEE International Conference on Communications Workshops (ICC Workshops), pages 1–6, May 2019.

[10] Mingxi Tan, Dazhi Sun, and Xiaohong Li. A secure and efficient blockchain-based key management scheme for lorawan. In 2021 IEEE Wireless Communications and Networking Conference (WCNC), pages 1–7, 2021.

[11] Syed Danish, Marios Lestas, Hassaan Qureshi, Kaiwen Zhang, Waqar Asif, and Muttaqirushan Rajarajan. Securing the lorawan join procedure using blockchains. Cluster Computing, 23, 09 2020.

[12] Victor Ribeiro, Raimir Holanda, Alex Ramos, and Joel JPC Rodrigues. A lightweight blockchain based two factor authentication mechanism for lorawan. In 2018 International Conference on Crowd Science, 1(3):270–280, 2017.

[13] Ben Buurman, Joarder Kamruzzaman, Gour Karmakar, and Syed Islam. Low-power wide-area networks: Design goals, architecture, suitability to use cases and research challenges. IEEE Access, 8:17179–17220, 2020.

[14] Usman Raza, Parag Kulkarni, and Mahesh Sooriyabandara. Low power wide area networks: An overview. IEEE Communications Surveys & Tutorials, 19(2):855–873, 2017.

[15] Claire Goursaud and Jean-Marie Gorce. Dedicated networks for iot: Phy/mac state of the art and challenges. EAI endorsed transactions on Internet of Things, 2015.

[16] Jonathan de Carvalho Silva, Joel JPC Rodrigues, Antonio M Alberti, Petar Stanchev, and Andre LL de Souza. Lorawan—a low power wan protocol for internet of things: A review and opportunities. In 2017 2nd International Multidisciplinary Conference on Computer and Energy Science (SplTech), pages 1–6. IEEE, 2017.
Luiz Oliveira et al. Performance assessment of lorawan and sigfox protocols in mobility scenarios, 2018.

LoRa Alliance. Lorawan® specification v1.1 | lora alliance®. https://lorawan.org/resource-hub/lorawan-specification-v11, 2018. (Accessed on 06/14/2021).

LoRa Alliance. Lorawan® back-end interfaces v1.0 | lora alliance®. https://lorawan.org/resource-hub/lorawan-back-end-interfaces-v1-0/, (Accessed on 06/14/2021).

Robert Miller. Lora security: Building a secure lora solution. MWR Labs Whitepaper, 2016.

Klaithern Al Nuaimi, Nader Mohamed, Mariam Al Nuaimi, and Jameela Al-Jaroodi. A survey of load balancing in cloud computing: Challenges and algorithms. In 2012 second symposium on network cloud computing and applications, pages 137–142. IEEE, 2012.

Einollah Jafarnejad Ghomi, Amir Masoud Rahmani, and Nooruldeen Nasih Qader. Load-balancing algorithms in cloud computing: A survey. Journal of Network and Computer Applications, 88:50–71, 2017.

Enrique Vargas and Sun BluePrints. High availability fundamentals. Sun Blueprints series, pages 1–7, 2000.

Zhibin Zheng, Shaona Xie, Hong-Ning Dai, Xiaping Chen, and Huaimin Wang. Blockchain challenges and opportunities: A survey. International Journal of Web and Grid Services, 14(4):352–375, 2018.

Murat Kuzu, Manisa Pipattanasomporn, Levent Gurses, and Saifur Rahman. Performance analysis of a hyperledger fabric blockchain framework: throughput, latency and scalability. In 2019 IEEE international conference on blockchain (Blockchain), pages 536–540. IEEE, 2019.

Carlos Melo, Felipe Oliveira, Jamilson Dantas, Jean Araujo, Paulo Pereira, Ronierison Maciel, and Paulo Maciel. Performance and availability evaluation of the blockchain platform hyperledger fabric. The Journal of Supercomputing, pages 1–23, 2022.

Suporn Pongnumkul, Chaiyaphum Siripanpornchana, and Suttipong Thachayapong. Performance analysis of private blockchain platforms in varying workloads. In 2017 26th International Conference on Computer Communication and Networks (ICCCN), pages 1–6. IEEE, 2017.

Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Bing Chen Ooi, and Kian-Lee Tan. Blockbench: A framework for analyzing private blockchains. In Proceedings of the 2017 ACM International Conference on Management of Data, pages 1085–1100, 2017.

J. Kreps, Neha Narkhede, Jun Rao, et al. Kafka: A distributed messaging system for log processing. In Proceedings of the NetDB, volume 11, pages 1–7, 2011.

Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In 2014 {USENIX} Annual Technical Conference ({USENIX} {ATC} 14), pages 305–319, 2014.

Chirpstack open-source lorawan® network server. https://www.chirpstack.io/, (Accessed on 06/18/2021).

Luca Foschini, Andrea Gavagna, Giuseppe Martusselli, and Rebecca Montanari. Hyperledger fabric blockchain: Chaincode performance analysis. In ICC 2020-2020 IEEE International Conference on Communications (ICC), pages 1–6. IEEE, 2020.

Mohamed Eldefrawy, Ismail Butun, Nuno Pereira, and Mikael Gilllund. Formal security analysis of lorawan. Computer Networks, 148:328–339, 2019.

Christian Cachin et al. Architecture of the hyperledger blockchain fabric. In Workshop on distributed cryptocurrencies and consensus ledgers, volume 310. Chicago, IL, 2016.

VICTOR RIBEIRO received the M.S. degree in applied computer science from the University of Fortaleza (UNIFOR), Fortaleza, Brazil, where he is currently pursuing the Ph.D. degree. His main research interests include the IoT, blockchain, and cloud security.

RAIMIR HOLANDA received the Ph.D. degree in computer engineering from Polytechnic University of Catalonia, Spain in 2005 and Master Science degree in computer science from Federal University of Ceará, Brazil in 1998. He did a Post-Doc at Université Pierre-et-Marie-Curie, France in 2020. Currently, Raimir Holanda is a Full Professor with the University of Fortaleza, Brazil.

ALEX RAMOS, JR. received the Ph.D. and master’s degrees in applied computer science from the University of Fortaleza (UNIFOR), Fortaleza, Brazil, in 2018 and 2013, respectively. He is currently a Professor with the Federal Institute of Education, Science and Technology of Ceará, Fortaleza.

JOEL J. P. C. RODRIGUES [AAIA and IEEE Fellow] is with College of Computer Science and Technology, China University of Petroleum (East China), China; Senac Faculty of Ceará, Brazil; and senior researcher at the Instituto de Telecomunicações, Portugal. He is the leader of the Next Generation Networks and Applications (NetGNA) research group (CNPq), Member Representative of the IEEE Communications Society on the IEEE Biometrics Council. He was Director for Conference Development - IEEE ComSoc Board of Governors, an IEEE Distinguished Lecturer, a Past-Chair of the IEEE ComSoc TCs on eHealth and on Communications Software, and Steering Committee member of the IEEE Life Sciences Technical Community. He is the editor-in-chief of the International Journal of E-Health and Medical Communications and editorial board member of several journals. He has authored or coauthored about 1000 papers in refereed international journals and conferences, 3 books, 2 patents, and 1 ITU-T Recommendation. Prof. Rodrigues is a member of the Internet Society, a senior member ACM, and a Fellow of AAIA and IEEE.

* * *