This is a preprint version of the paper entitled "A Cloud-Based Key Rolling Technique for Alleviating Join Procedure Replay Attacks in LoRaWAN-based Wireless Sensor Networks". The published version is available in IEEE Xplore:https://ieeexplore.ieee.org/document/10825339

A Cloud-Based Key Rolling Technique for Alleviating Join Procedure Replay Attacks in LoRaWAN-based Wireless Sensor Networks

Dimitra Papatsaroucha
Department of Computer & Electrical Engineering
Hellenic Mediterranean University
Heraklion, Greece
dpapatsa@hmu.gr

Nikolaos Astyrakakis
Department of Computer & Electrical Engineering
Hellenic Mediterranean University
Heraklion, Greece
n.astyrakakis@pasiphae.hmu.gr

Evangelos Pallis
Department of Industrial Design and Production Engineering
University of West Attica
Athens, Greece
epallis@uniwa.gr

Panagiotis I. Radoglou Grammatikis
[1] K3Y, Studentski district
Sofia, Bulgaria
[2] Department of Informatics and Telecommunication Engineering
University of Western Macedonia
Kozani, Greece
pradoglou@k3y.bg,
pradoglou@uowm.gr

Panagiotis G. Sarigiannidis
Department of Informatics and Telecommunication Engineering
University of Western Macedonia
Kozani, Greece
psarigiannidis@uowm.gr

Evangelos K. Markakis
Department of Computer & Electrical Engineering
Hellenic Mediterranean University
Heraklion, Greece
emarkakis@hmu.gr

Abstract*— Nowadays, numerous devices are utilizing the IoT world, connecting and providing access to data and sensor measurements in vast networks of interconnected objects and devices. Considering the great communication distances that need to be covered occasionally, the LoRaWAN network was proposed as it employs Low Power (LP) and Long Range (LoRa) protocols that reduce device energy consumption while maximizing communication range. A gateway to the cloud authenticates LoRaWAN IoT devices before data transmission. This procedure begins with an unencrypted Join Request. A Join Request includes, among others, a Message Integrity Code (MIC), which is the result of encrypting the unencrypted contents of the message using an AppKey that is securely stored both in the cloud and the IoT device. However, malicious actors acting as Man-In-the-Middle (MITM) can interfere in the communication channel, reverse engineer the MIC value, and derive the AppKey. They can then initiate a Join Request that is misinterpreted as coming from a legitimate device and gain access to the communication channel. This paper introduces a novel approach that focuses on the continuous regeneration of the AppKey, necessitating frequent re-joining and re-authentication of IoT devices within the network. The suggested method, which can be added as an extra layer of security in LoRaWAN networks, uses a key rolling technique similar to the one used in automobile central locking systems, and is developed as an optimised and scalable microservice for various LoRaWAN installations and versions. Through the evaluation process, significant findings emerged, demonstrating the effectiveness of the proposed security solution in mitigating replay attacks. The system successfully prevented the server from getting flooded by malicious packets, distinguishing it from a system lacking the proposed mechanism. Remarkably, this accomplishment was made without causing any noticeable delay to the communication process. In addition, the timeframe required by the proposed mechanism to generate the new AppKey is assumed to be too short for attackers to execute a replay attack, considering the computational resources currently accessible.

***Keywords*— AppKey, Internet of things, Key Rolling, LoRaWAN Security, Join Procedure, Rolling Code**

I. INTRODUCTION

In the newest technological era, the Internet of Things (IoT) [1] is constantly expanding, enabling the use of smart devices in most communities and areas, from smart homes to smart cities, fields, and more. These devices are designed for indoor and outdoor use and they can transmit data across distances of several kilometers. Meanwhile, the use of specialized protocols for regulating data flows and device sleep intervals allows some of these devices to operate with reduced battery consumption, depending on the use case and configuration.

Low Power Wide Area Networks (LPWAN), such as LoRaWAN[2], SigFox[3], and NB-IoT[4], are specialized for long-distance data transmissions. These networks are usually employed in agriculture, manufacturing, logistics, and a variety of other outdoor and indoor applications for sensor data transmission, which may also include sensitive or personal data [1]. Nevertheless, the research community raises concern over the security techniques and processes now utilized in these networks. This is mostly owing to the inherent vulnerability of the air as a medium for data transfer. In addition, according to literature, LoRaWAN networks have been found to have insufficient security measures during the Join Procedure, which is the process of authenticating devices that are part of these networks [2], [3]. The initial Join Request message sent by the device to the cloud includes a Message Integrity Code (MIC), which is used to ensure the integrity of the Join Request. The device utilizes a key named AppKey to generate the MIC value by encrypting the contents of the Join Request using the AES-128 encryption method. Additionally, the cloud stores and employs the same AppKey to decipher the MIC value and verify the authenticity of the device when a Join Request is received. It is worth mentioning that the

[1] https://csrc.nist.gov/glossary/term/internet_of_things
[2] https://lora-alliance.org/about-lorawan
[3] https://www.sigfox.com/en
[4] https://www.gsma.com/iot/narrow-band-internet-of-things-nb-iot

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101070455 (DYNABIC). Disclaimer: Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or European Commission. Neither the European Union nor the European Commission can be held responsible for them.

static nature of the AppKey in LoRaWAN makes it susceptible to cyber-attacks, such as replay attacks. Research has demonstrated that attackers may reverse engineer the MIC value, derive the AppKey, and get access to the communication channel by impersonating a legitimate device [3], [4].

This paper introduces a new method for enhancing the security of LoRaWAN networks by enhancing the authentication process of devices. It suggests adding an extra layer of security on top of the Join Procedure of the Over The Air Authentication (OTAA) mode, as outlined in both the 1.0.2 and the most recent 1.1 versions of the LoRaWAN specification document. By integrating a key rolling approach, also known as rolling code, with LoRaWAN networks the proposed implementation puts forward a solution towards counteracting replay attacks without any additional complexity or delay induced to the communication. The key rolling approach replaces the static AppKey used for device authentication with a dynamic AppKey, which changes before each message exchange session, requiring the device to re-join the network. The dynamic AppKey is consistently transferred to the device through a secure tunnel using session keys, making it difficult for attackers to interfere with the communication channel, as an adversary would need significant computational resources to intercept and decipher the MIC value in order to deduce the AppKey, before the AppKey is modified again. Although the rolling code method is vulnerable to replay attacks [5], [6], in LoRaWAN networks the use of multiple gateways over long distances can mitigate this issue by diminishing the ability of attackers to jam or restrict access to all gateways simultaneously.

The proposed implementation was tested in a real-world environment using PyCom FiPy Micro-controllers, Lorix One devices, and a private self-hosted Chirpstack framework. The results showed that this novel technique enhances the security of LoRaWAN networks by bolstering the device authentication procedure, preventing malicious users from manipulating data and compromising critical infrastructures. The solution does not alter the underlying infrastructure or LoRaWAN protocol, but simply adds two extra packets during successful data transmission to ensure device re-authentication.

The rest of this paper is structured as follows: the Background Theory section describes the main background theory of the device authentication procedure in LoRaWAN networks as well as the security limitations and vulnerabilities of this procedure; the Literature Review section presents state-of-the-art endeavors and techniques used to secure the Join Procedure of the OTAA mode in LoRaWAN networks; the Architecture section provides the proposed system architecture is provided alongside a description about each system component; the Implementation showcases the implementation of the proposed key rolling technique; the Evaluation section reports the results after the evaluation of the proposed system in a real-world environment; and the Conclusion section presents concluding remarks and proposed future steps.

II. Background Theory

The authentication of devices participating to LoRaWAN networks as well as the security limitations and vulnerabilities of this procedure are briefly yet concisely described in this section. This paper takes into consideration the 1.0.2 version of the LoRaWAN specification [5] ; however, most of the concepts, limitations, and vulnerabilities described in the following subsections apply also to the latest versions of the LoRaWAN specification (e.g., 1.1).

A. Authentication of Devices Participating in LoRaWAN networks

As per the existing security measures of the LoRaWAN network and the versions 1.0.2 and 1.1 of the LoRaWAN specification, LoRaWAN networks requires extreme edge IoT devices (LoRaWAN clients) to undergo an activation and registration process to join and operate within the network. This can be achieved through the OTAA security verification procedure or the Activation By Personalization (ABP) method[6]. In OTAA mode, devices transmit unencrypted Join Requests to the cloud for verification, referred to as the Join Procedure. In contrast, ABP uses a device-specific mechanism for activation, which is beyond this research's scope.

A Join Request consists of two identifiers (AppEUI and DevEUI), a random message value (DevNonce), a Media Access Control (MAC) header (MHDR), and the MIC value. The AppEUI is a universal identifier used for Join server identification while DevEUI is a 64-bit universally unique number used for device identification. The DevNonce is a randomly generated number to prevent replay attacks. The cloud records DevNonce values used in Join Requests and denies requests with existing DevNonce values. Furthermore, the MAC header specifies the message type and primary version of the frame format employed by the LoRaWAN layer specification for encoding the frame. The MIC value is obtained by encrypting the AppEUI, DevEUI, DevNonce, MHDR, or a combination of them using a cryptographic key called AppKey.

The AppKey is created in a random manner and saved in the cloud for each extreme edge device that requires communication with the cloud servers. It is also manually entered into each extreme edge device. When a Join Request is delivered to the cloud, the AppKey is used to decrypt the MIC value and so authenticate the extreme edge device that issued the Join Request. The cloud then uses the AppKey to generate two session keys (AppSKey and NwkSKey) by encrypting either the DevNonce or a value named AppNonce. The AppNonce is a randomly generated value or a predetermined unique ID in the cloud that is communicated from the cloud to the extreme edge device as part of the Join Procedure's encrypted Join Accept message. The extreme edge device then uses the AppNonce in conjunction with the AppKey to generate exactly the same AppSKey and NWkSkey session keys as those stored in the cloud. Following that, both the extreme edge device and the cloud use these session keys to encrypt and verify network communication and application data throughout their remainder of the communication. The contents of the Join Accept message are outside of the scope of this research since, unlike Join Request messages, Join Accept messages are encrypted before

[5] https://lora-alliance.org/resource_hub/lorawan-specification-v1-0-2
[6] https://www.thethingsindustries.com/docs/devices/abp-vs-otaa

transmission. The LoRaWAN Join Procedure of the OTAA mode is illustrated in detail in Fig 1.

B. Security Limitations and Vulnerabilities of the Join Procedure in LoRaWAN networks

While LoRaWAN networks adhere to the specifications set by the LoRa Alliance and are deemed secure, the security of these networks is a subject of debate in the relevant literature. As highlighted in [2], it has been observed that LoRaWAN networks operating in OTAA mode do not implement adequate security measures during the Join Procedure.

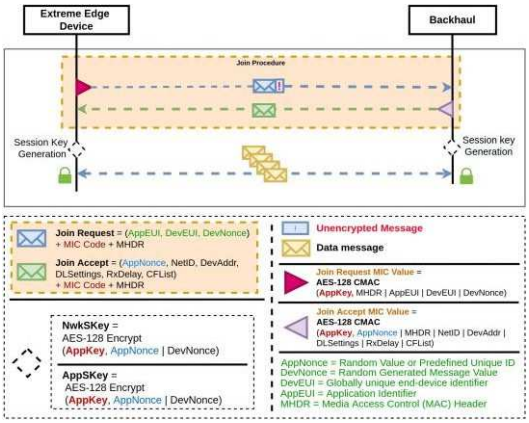

Fig 1. LoRaWAN OTAA Join Procedure essentials

The MIC value transmitted over LoRaWAN networks is generated using the AES-CMAC algorithm, which employs AES-128 blocks for encryption that are highly resistant to cracking. However, Isa H. et al in [4] have outlined various algorithms and techniques that could potentially decrease the processing power required to decrypt AES-128 messages more quickly. In addition, the AppEUI and DevEUI identifiers, the DevNonce value, and the MHDR header are transmitted unencrypted over the air, as part of a Join Request message, even in the latest 1.1 version of the LoRaWAN specification. Furthermore, these values stay constant for every Join Request transmitted by a particular edge device, in addition to the AppKey, which is likewise a static, one-time generated key. Furthermore, studies have demonstrated that the DevNonce value is susceptible to interception using a brute force attack [3].

Consequently, devices utilizing LoRaWAN networks over long distances are susceptible to eavesdropping and replay attacks [7] , as also discussed in [3]. An adversary could potentially disrupt the communication channel, attempt to extract the AppKey by reverse engineering the MIC value, and send a Join Request that is misinterpreted by the cloud as coming from a legitimate device. This could lead to the manipulation of data by imitating the actions of the targeted device, which is commonly referred to as an impersonation attack. Thereby, a malicious actor may manipulate and store tampered data in the cloud, which could potentially resulting in severe repercussions, especially in critical infrastructure like healthcare or pollution monitoring, as discussed in [7].

III. Literature Review

In this section, this paper delves into the current landscape surrounding security concerns in LoRaWAN networks as well into the different types of attacks that can target these networks and the most recent efforts to mitigate these risks. In addition, this section explores recent research on the application of the rolling code technique and the effectiveness of AES 128bit encryption keys in ensuring security.

A. Roling Code Technique

The rolling code technique is a well-established and frequently used technology in remote controlled security systems. The vehicle lock systems, garage door opener systems, industrial systems and more, utilize such technology to prevent unauthorized access in their premises or systems.

A literature review performed by Lukasz Migacz et. al in [8] analyzed all vehicle security systems, from 1949 to 2021. The authors presented various security systems that utilize the rolling code technique, such as Keyloq, DST-40, HiTag, etc. An analysis of HiTag 2 Integrated Circuit (IC) conducted by Flavio D. Garcia et al. in [6] resulted in cracking the HiTag 2 IC algorithm, while an investigation conducted by Ahmed Ghanem, et. al in [9] led to the development of a new algorithm for cracking various garage door openers that utilize a rolling code technique. However, in the first scenario, the HiTag 2 IC algorithm employed a 48-bit key and a timer set to 360 seconds for protection. Similarly, in the second scenario, the rolling code techniques utilized small payloads or encryption keys that were less than 128 bits. Since 1949, as indicated in the survey, the implementation of 128-bit encryption and the use of timers are employed to mitigate attacks and enhance the security level of the rolling code technique, while no other substantial changes have been observed.

It is important to mention that, to this day, vehicle locking systems continue to employ rolling code systems enhanced with 128-bit encryption keys, which, to the best of our knowledge, have not been compromised. Cracking such systems using 128-bit encryption schemes necessitates a significant amount of computational power, which is presently inaccessible. In addition, in order to bolster security measures, developers may employ a variant of the rolling code technique that leverages the EnOcean protocol, as suggested by Katharina Hofer-Schmitz in [10]. This method is proposed to thwart potential threats such as system hijacking, jamming, and replay attacks.

B. LoRaWAN

The increasing number of IoT networks has sparked research on their challenges, particularly in LoRaWAN networks, which, as part of the IoT networks, have the capability to transmit crucial data within industrial infrastructures to facilitate automated sequential operations, or they can be utilized to convey personal data within residential infrastructures. Cybersecurity experts have identified unresolved issues and strategies to address them. This section examines recent advancements in LoRaWAN network security.

Kim J. and Song J. in [11] presented an authentication technique that substitutes the conventional Join Procedure with two procedures. The first procedure, known as the Initial Join Procedure, aligns with the Join Procedure outlined by the LoRa Alliance in terms of device authentication. In addition, the authors have developed a procedure called "Non-Initial Join Procedure" to enhance the security of LoRaWAN

[7] https://csrc.nist.gov/glossary/term/replay_attack

networks. This procedure focuses on performing a secondary custom device authentication. Nevertheless, the Join Request contents and the AppKey of the Initial Join Procedure lack encryption and remain unchanged, which poses a significant risk to the entire network.

SeungJae Na, et al. in [2], also identified the vulnerability of the Join Procedure and suggested a method for concealing the DevNonce and MIC values of the Join Request through masking. Masking is the process of concealing a series of characters or numbers while preserving the original format. This is achieved by employing a masking token, such as a string or integer, in combination with a logical operation like XOR or AND, which are commonly employed in cryptography. The suggested approach retrieves the first masking token from the AppKey and employs it to mask the DevNonce and MIC values of the Join Request through AES-128 encryption. Subsequently, the NwkSKey from each data transmission session is employed to obfuscate the DevNonce and MIC values of the subsequent transmission session.

In a similar manner, Thomas J. et al. in [12] introduced a modified masking algorithm aimed at preventing the retrieval of the AppKey through the decryption of the MIC value. Their method utilizes a modified cryptography counter mode technique, which differs from the conventional block cipher mode (CTR) used in AES-128 encryption. They similarly suggested masking the DevNonce and MIC values of the Join Request using the AppKey while using the NwkSKey of the previous data transmission session to mask the DevNonce and MIC values of the subsequent data transmission session. It is worth noting, that the overall safety of the both proposed systems is contingent upon the attacker's expertise, capabilities, and computational resources, as well as the information gathered during reconnaissance prior to the attack. An adversary coule intercept a Join Request message, decipher the masked MIC value to obtain the AppKey, and then replicate a valid Join Request. Then, when connecting to the network, the attacker could get the NwkSKey for each data transmission session and employ it to obfuscate the DevNonce and MIC values of the subsequent transmission session. Consequently, both systems remain susceptible to exploitation by malicious actors, who may replicate this series of operations and take advantage of the LoRaWAN network.

On another note, Ntshabele et al. in [13] suggested implementing a proactive time-dependent protection mechanism to counteract replay attacks on LoRaWAN networks. To ensure this, they incorporate a timestamp into every message transmitted during request-to-join sessions. Each message's timestamp is stored and verified upon reception. If an attacker tries to transmit a tampered message, there can be a difference between the timestamp added to the message by the attacker and the expected timestamp by the server due to the time it takes for replication and transmission. If the timestamps do not align, the join server immediately rejects the message and notifies the user of the discovery of a replay attack. Similarly, Hayati et al. [14] suggested a method to include timestamps, derived from the MIC value, into the sent Join Request messages, while also containing nonce counters. Upon receiving the request messages, the join server assesses the timestamps and nonce counters. Unlike the previous method, if the attacker is able to successfully change the timestamp, the message is still verified using the MIC value that was originally used to produce the timestamp. Both approaches significantly enhance security by promptly notifying users of any deviation in the received timestamps compared to the transmitting entity's prior timestamp, indicating a possible occurrence of a replay attack. Nevertheless, they fail to deter the replay attack from abusing the network and enabling malicious individuals to gain unauthorized access to potentially critical data or infrastructures.

An alternative method for securing the Join Procedure was proposed by Sung, W. J. et al in [15]. Their proposed method relies on the cloud and utilizes the Received Signal Strength Indicator (RSSI) that shows how far a device may be depending on its signal strength. Moreover, a handshaking technique alongside proprietary messages are used to ensure integrity of messages and prevent replay attacks. However, this solution lacks the validation of the RSSI value around a gateway; therefore, an attacker being in the same distance as a legitimate device and having the same RSSI may be misinterpreted as legitimate device.

Furthermore, Xia, Z. et al. in [16] proposed a Key Distribution Server (KDS) that introduces a decentralized key provision infrastructure for securing the Join Procedure of LoRaWAN. This infrastructure generates dynamic session keys for the extreme edge LoRaWAN devices, with a specific expiration date, and performs a double Join Procedure: (1) a Join Request is sent from the extreme edge device to the KDS, (2) the KDS validates the device and sends a second Join Request to the network server, (3) a Join Accept is sent from the network server to the KDS and (4) a second Join Accept is sent from the KDS to the extreme edge device. This KDS infrastructure partially resolved the Join Procedure vulnerability of LoRaWAN networks, due to the update of the session keys after their expiration and the double authentication procedure; however, at the expense of increased network complexity, packet size, and latency.

Similarly, Pathak et al. [17] proposed a centralized session key mechanism aimed at mitigating replay attacks within LPWAN, which is closely related to the LoRaWAN standard. Their approach, which included timestamps in session beginning packets, was also intended to verify the freshness of delivered messages. The methodology foresaw potential points of failure and recommended the use of redundant servers in future configurations - a crucial foresight in the rapidly evolving field of IoT security.

Furthermore, Hayati et al. [18] described an approach for upgrading and altering the "root" key, known as NwkKey, compared to the AppKey change proposed in this paper. Their technique, which featured timestamps and nonce count procedures, was designed not only to withstand replay attacks, but also to promote cost-effective communication through the use of session-specific keys. However, as the authors point out, the proposed technique modifies the LoRaWAN protocol and it is not easily implemented in real-world IoT scenarios.

Finally, Lin, J. et al. in [19] and Danish, S. M. et al. in [20] presented a technique for securing LoRaWAN networks and ensuring packet integrity using the blockchain concept. In [19] the authors proposed a blockchain schema for constructing a secure LoRaWAN network, using the blockchain architecture in the cloud, and ensuring data integrity for all received messages. Furthermore, in [20] a blockchain two-factor authentication module was proposed, which can be enabled on the gateway of any LoRaWAN network. In the latter case, the gateway is responsible for verifying the integrity of received

packets with the Ethereum blockchain before transporting the packets to the cloud. Although these methods are effective for safeguarding LoRaWAN networks, the authors noted that physical attacks might compromise the blockchain keys and, as a result, the entire security mechanism upon which LoRaWAN networks rely.

To summarize the findings from the literature research, the majority of the suggested methods alter the LoRaWAN protocol to boost security during the Join Procedure in LoRaWAN networks, in the expense of reducing backwards compatibility. Furthermore, the majority of the examined suggested solutions maintain static keys throughout the process, meaning that the effectiveness of these solutions relies on the attacker's knowledge, skills, and computational resources to extract the AppKey with reverse engineering. In contrast, the proposed solution in this paper is a lightweight implementation that adds an extra layer of security to LoRaWAN networks. It does not introduce any additional complexity, increase packet size, or cause latency as it was observed during its evaluation. Furthermore, it does not require any modifications to the existing LoRaWAN protocol, making it compatible with both previous and future versions of the LoRaWAN specification.

In addition, the dynamic alteration of the AppKey, based on the rolling code technique, and its transmission to the end device through a secure tunnel, as suggested in this paper, indicates that if an attacker manages to obtain the AppKey from an intercepted message, by the time they send a tampered Join Request to the server, the AppKey is no longer valid and thus their message is rejected. This not only assists in recognizing replay attacks, as described in previously mentioned literature, but also may prevent severe impact from such attacks in critical infrastructures.

Last but not least, the utilization of a rolling code approach to modify the AppKey in LoRaWAN networks benefits from the several gateways that are employed over extensive distances in these networks. This makes it challenging for attackers to simultaneously jam signals to all gateways and capture the Join Request message, before it arrives to any gateway and the procedure of dynamic AppKey regeneration begins.

IV. Architecture

This section delves into the details of the system architecture that underpins the proposed security mechanism.

A. Infrastructure Components

The system architecture diagram in Fig 2 illustrates the infrastructure components of the LoRaWAN network, the key rolling module, and the interconnection protocols/interfaces used by the following layers: the edge devices (i.e., LoRaWAN gateways), the extreme edge devices (i.e., LoRaWAN clients), and the cloud infrastructure (i.e., Chirpstack network/application servers). An example end node device with the AppKey saved in flash memory is represented on the left side of the system architecture diagram, indicated with yellow color. Every time this end node device needs to join the LoRaWAN network, it transmits the AppKey over the air in plain text format, which malicious users might exploit as explained in Introduction and Background Theory Sections of this paper.

The Rolling Key Module, which represents the security mechanism proposed in this research, may be seen on the right side of this diagram. On each new message received from an end device, this module replaces the AppKey with a newly generated random key. This method prohibits re-transmissions of the same key over the air and can actively prevent malicious users, as detailed in the sections that follow that discuss its functionality and implementation. The key is then sent to the extreme edge device over a secure tunnel opened using session keys. The New AppKey is saved in the device's flash memory (NAND) and the retrieval is acknowledged. The Rolling Module then modifies the AppKey associated with the device in the application server's database.

The conceptual diagram in Fig 3 shows the proposed system to protect LoRaWAN networks from MITM attackers. Two extreme edge LoRa devices are depicted, where: the one at the top is protected using the proposed key rolling technique and a new AppKey for each communication; the one at the bottom is unsecured as it uses a static AppKey. At the center of the diagram, a potential attacker is presented acting as a MITM and attempting to perform a replay attack and gain access to both secured and unsecured devices. The gateway, presented also at the center, collects communications and forwards them to the cloud, while the Chirpstack architecture, which can be seen at the right side, provides the application and network servers to which the gateway communicates via a secure TCP connection over SSL certificates. At the same side of the diagram, the Key Rolling Module is showcased operating as a cloud native application listening for messages arriving at the Chirpstack framework and initiating particular actions.

B. Network Protocols and Connectivity Details

The suggested solution operates as a microservice that uses a WebSocket connection to monitor a specific device identified by its DevEUI. Additionally, it operates in conjunction with the key rolling module to update the AppKey with each new communication from this particular device. Each device is governed by a specific microservice that is dynamically allocated and relies on the following inputs, which are provided as environment variables: the DevEUI identifier of the device, the current AppKey, and the Fully Qualified Domain Name (FQDN) or IP Address of the cloud application server (i.e., the Chirpstack Framework) where the application data is recorded.

The microservice can be implemented either inside the same local area network (LAN) as the Chirpstack framework or on a separate network/subnet, as long as it has connectivity to the Chirpstack framework, such as through FQDN or a Public IP Address. If the Chirpstack framework is implemented on a public cloud infrastructure, the device can establish communication with it over the Internet. This illustrates the versatility of the proposed security solution, as it can be implemented in many scenarios, whether they include remote or local deployment, and may be applied to one or several devices concurrently.

In the context of remote deployment, it is necessary for the Chirpstack framework to possess reliable SSL certificates and a fully qualified domain name (FQDN) that can be accessed publicly. As seen in the conceptual diagram in Figure \ref{conceptual}, the cloud native application (i.e., key rolling module) transfers each newly generated AppKey to both the application server and the LoRa device. The transmission occurs over the secure RESTful API interface provided by the Chirpstack framework and the gateway, including the appropriate downlink message for each entity.

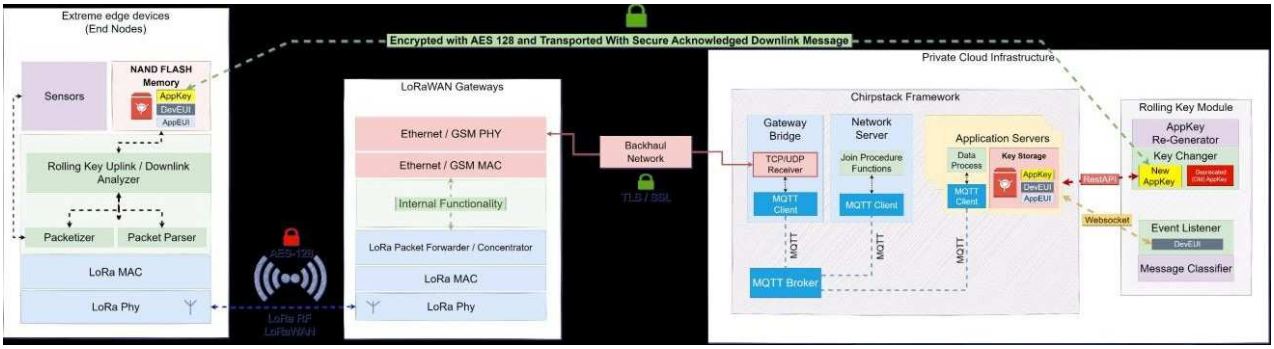

Fig 2. System Architecture with key rolling mechanism.

In the context of remote deployment, it is necessary for the Chirpstack framework to possess reliable SSL certificates and a fully qualified domain name (FQDN) that can be accessed publicly. As seen in the conceptual diagram in Figure \ref{conceptual}, the cloud native application (i.e., key rolling module) transfers each newly generated AppKey to both the application server and the LoRa device. The transmission occurs over the secure RESTful API interface provided by the Chirpstack framework and the gateway, including the appropriate downlink message for each entity.

V. Implementation

Before implementing the suggested solution, certain assumptions were needed to be made. The AppEUI IDs of all extreme edge devices were assigned as "0000000000000000" due to the Chirpstack framework's lack of requirement for an AppEUI in LoRaWAN versions 1.0.2 and/or higher. Even if an AppEUI is specified, the framework disregards its value[8]. In addition, in this implementation, the LoRaWAN standard version 1.0.2 was employed because the LoRa extreme edge devices and gateway being used do not support higher versions of LoRaWAN networks, such as 1.1. Nevertheless, the suggested dynamic key rolling technique in this paper can be modified to suit future versions of LoRaWAN networks. The suggested approach not only enhances the security of the Join procedure, but also provides an additional layer of security, even in the event that the vulnerability in the Join procedure is resolved in the future.

A. Description of Implementation

This paper uses a privately installed Chirpstack framework and PyCom devices to construct the proof-of-concept solution, with the aim to demonstrate the key rolling concept in LoRaWAN networks. The Chirpstack framework comprises several distinct components that need to be installed separately: the application server that is responsible for storing the device data and enabling the security techniques of LoRaWAN; the network server, which is responsible for the network communication and the Network layer security; an MQTT broker, named "Mosquitto," which facilitates the exchange of messages across different components within the Chirpstack framework and transfers the LoRaWAN packets in the MQTT message format from one component to another; a database of user's choice (e.g., PostgreSQL) for storing the aforementioned application and extreme edge device data.

In this implementation, the Chirpstack framework serves as the cloud, acting as both the application and the network server, and is accessible through a user interface and multiple sockets (e.g., Websockets). Extreme edge devices provide data to the Chirpstack framework, which may execute a range of activities. For instance, a device may potentially transmit temperature measurements at a frequency of one per minute. A cloud function or a cloud integration of the Chirpstack framework could be employed to trigger an action on another device participating in the network, such as opening an electric water valve to release water for plants. This particular examples portrays a real-wolrd use case scenario and a potential vulnerability in a critical industry infrastructure, where a malicious actor with the ability to eavesdrop, intercept, modify, or replay data may carry out unauthorized activities on cloud-connected devices. The key rolling module developed in this paper, as a cloud native application, functions by remotely operating with the Chirpstack framework. It achieves this by utilizing a RESTful API and WebSockets communications via TLS, along with service account credentials.

The Chirpstack framework's RESTful API interface was used to remotely control the Chirpstack framework. Following thorough investigation and experimentation with the RESTful API, system access was achieved into the Chirpstack framework, specifically the application server, using administrator user credentials and a token. The static AppKey was successfully replaced with a newly generated key, marking the completion of the initial phase of the proposed implementation. The Postman tool/software[9] was utilized to conduct every single experiment and test, enabling the exploration of the capabilities of the Chirpstack framework's RESTful API through HTTP/HTTPS requests.

Upon uncovering the method to modify the AppKey on the server, it became necessary to perform the same task on the device, remotely, using LoRaWAN packets. The objective was to transmit a downlink acknowledged packet containing a new key from the Chirpstack framework application server. This transmission would occur through the secure channel of the downlink, which is already encrypted using the NwkSKey and AppSKey obtained during the initial Join Procedure. Thus, the device solely retains knowledge of the new AppKey, in addition to the Chirpstack framework application server that transmits it. Upon the arrival of the key, the previous AppKey was substituted and the new one was effectively stored in the device's storage, specifically the NAND flash memory of PyCom devices. Subsequently, the device

[8] https://forum.chirpstack.io/t/where-do-i-set-the-appeui/381/3

[9] https://www.postman.com/

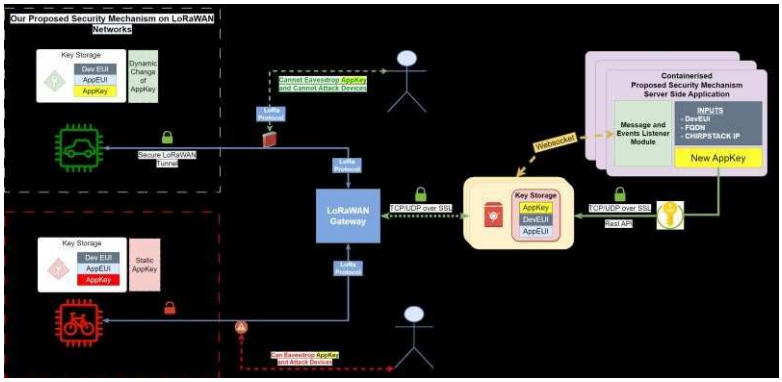

Fig 3. Conceptual Diagram

transmitted an acknowledgment of the alteration to the Chirpstack framework application server. The test concluded once the server effectively modified its AppKey after receiving the acknowledgement.

B. Flow Chart and Sequence Diagrams

The flow diagram depicted in Fig 4 illustrates the procedure undertaken by the Key Rolling Module to modify the AppKey. The process commences by activating the Key Rolling Module as depicted at the top of the diagram (Start). This microservice can be activated for a particular device by providing device-specific settings through environment variables when the container is started. Following the initialization process, the Key Rolling Module actively monitors incoming messages transmitted to the network server located in the cloud through the RestAPI interface and a WebSocket that are offered by the Chirpstack framework. Upon the arrival of a message to the cloud infrastructure, the Key Rolling Module verifies the "Type" of the message. When the message type is "UP", the Key Rolling Module starts the process of changing the AppKey for the designated device. The system creates the new AppKey and transmits it to the extreme edge device through a downlink message that requires acknowledgment. Once the extreme edge device confirms receipt of the message and stores it in its flash memory, the Key Rolling Module proceeds to store the newly generated AppKey in the application server database.

It is important to note that when a new AppKey is saved for a device in the database, it automatically renders the previous AppKey invalid. Once all the aforementioned steps are completed, the Key Rolling Module will restart the entire process. It will then wait for another "UP" message type to initiate the key change procedure again.

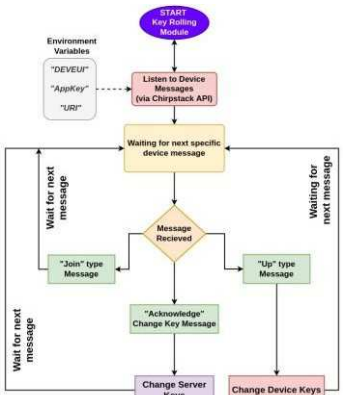

Fig 4. Flow Chart Diagram

Following, the sequence diagram depicted in Fig 5 showcases the flow of information within the proposed solution and highlights the security improvement that results from utilizing the rolling code technique in LoRaWAN networks. The LoRaWAN network components used in this diagram are from left to right: the extreme edge devices (LoRaWAN clients), the edge device (gateway), the cloud (Chirpstack Framework), and the Key Rolling Module proposed in this paper.

The first iteration of the communication begins with an unencrypted Join Request that is transmitted through the air. The Join Accept is subsequently transmitted to the extreme edge device through a series of authentication procedures carried out by the network server. Simultaneously, the key rolling module remains in a state of readiness to receive new messages from the device and generates a new AppKey for every message received.

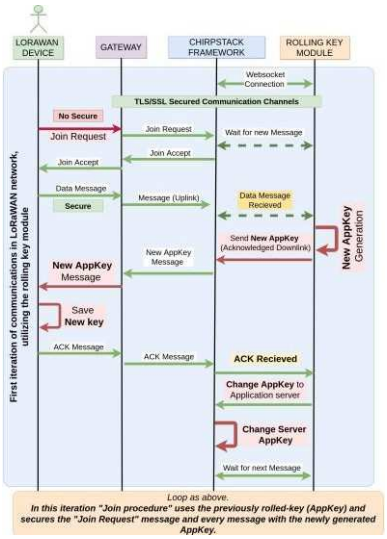

Fig 5. Sequence Diagram

C. Pseudo-codes

1) Server-side

The pseudo-code depicted in Fig 6 pertains to the server (cloud) aspect of the proposed security mechanism. Initially, a connection is established with the cloud-based application "Database" (e.g., Chirpstack Application Server), followed by the system awaiting incoming messages from devices. Upon the arrival of a message, the code verifies the validity of the AppKey and determines whether the incoming message is

associated with a Join request. Provided that all requirements are satisfied, a new random hexadecimal AppKey is generated and transmitted to the device that sent the Uplink message. The transmission is accompanied by an encrypted downlink-acknowledged message, which is encrypted using session keys.

```
DATABASE_CLIENT = CONNECT_TO_DB()

MESSAGE_1 = WAIT_FOR_UPLINK()->MESSAGE

IF VERIFY_APPKEY(MESSAGE_1->appKey, MESSAGE_1->device_id) == "VALID";
THEN
    IF MESSAGE_1->type == 'JOIN_MESSAGE';
    THEN
        DOWNLINK("JOIN_APPROVED_MESSAGE");
        NEW_RANDOM_KEY = RANDOM_HEX();
        DOWNLINK(NEW_RANDOM_KEY);
        MESSAGE_2 = WAIT_FOR_UPLINK()->MESSAGE;
        if MESSAGE_2->type == "ACK_MESSAGE";
        THEN
            DATABASE_CLIENT->SAVE_NEW_APPKEY_TO_DB_FOR_DEVICE(UPLINK_MSG->device_id, NEW_RANDOM_KEY);
        ENDIF
    ENDIF
ELSE
    DOWNLINK("JOIN_REJECT_MESSAGE");
ENDIF


function VERIFY_APPKEY(received_appkey, device_id):
    db_saved_appkey = DATABASE_CLIENT->GET_APPKEY_FROM_DB_FOR_DEVICE(device_id)
    IF db_saved_appkey == received_appkey;
    THEN
        return "VALID";
    ELSE
        return "INVALID";
    ENDIF
```

Fig 6. Server-side Pseudo-code

Subsequently, the server awaits confirmation from the device. Upon receiving the acknowledgment message, the server determines that the device has successfully received the new AppKey through the secure session tunnel. Consequently, the server proceeds to store the new AppKey in the application server database. Then, it remains in a state of waiting for the next message, and the process is repeated. If an invalid AppKey is sent from the client device during the Join Procedure, the Join request message is rejected in accordance with the specifications of the LoRa Alliance[10]

2) Device-side

The pseudo-code depicted in Fig 7 corresponds to the client-side of the proposed security mechanism. At the outset, the device initiates its boot process and executes all the required initialization commands as specified in the boot-loader.

```
INIT...
BOOT...
WHILE(TRUE):
    keys_from_flash = READ_FLASH_KEYS();
    measurements = GET_DEVICE_MEASUREMENTS();
    received_join_reply =  PLINK("JOIN_MESSAGE", keys_from_flash);
    IF (received_join_reply == "ACKNOWLEDGE");
    THEN
        ...
        # Session keys (NwkSKey and AppSKey) are generated with the normal proccedure defined in LoRaWAN specification.
        ...
        session_keys = GENERATE_SESSION_KEYS(NwkSKey, AppSKey);
        ...
    ENDIF

    IF(session_keys -> EXISTS && measurements -> EXISTS);
    THEN
        measurements_successfully_send = UPLINK(measurements, session_keys);
        IF measurements_successfully_send == TRUE;
        THEN
            new_appkey_message = WAIT_FOR_DOWNLINK()->message;
            IF(new_appkey_message->type == 'NEW_APPKEY' AND new_appkey_message->data == VALID_NEW_APPKEY_IN_HEX_FORMAT());
            THEN
                uplink_successfully_send = UPLINK('NEW_APPKEY_RECEIVED');
                IF uplink_successfully_send == TRUE;
                THEN
                    WRITE_FLASH_KEYS(NEW_APPKEY->data);
                ENDIF
            ENDIF
        ENDIF
    ENDIF
    SLEEP(xxx); #seconds
```

Fig 7. Device-side Pseudo-code

Subsequently, it enters into an endless cycle of transmitting data to the cloud. The device retrieves the predefined keys from its flash memory and thereafter retrieves the data that will be transmitted to the cloud. Afterwards, the device initiates a request to connect to the LoRaWAN network using the predefined keys. Upon receiving approval, the device generates Session Keys (NwkSKey and AppSKey) using the standardized LoRaWAN procedure outlined in the specification of the LoRa Alliance[11].

Furthermore, once the session keys are generated and a connection is established with the LoRaWAN Gateway and Cloud Infrastructure, the device transmits the data as expected. After sending the data, the device is waiting for a new valid AppKey from the server (cloud) side. Once the AppKey is received, the device promptly sends a "Acknowledge" to the server using an uplink-acknowledge message. Additionally, it stores the new AppKey in the flash memory. Ultimately, the device enters a state of inactivity for a specific duration of time, as such devices normally do, prior to transmitting subsequent data messages which will lead to the next round of AppKey alteration.

VI. Evaluation

The evaluation of the suggested system involved conducting three tests, specifically the Alpha and Beta experiments, as well as the Delay Experiment. The assessment of the suggested solution occurred in a real-world operational setting, using an on-premise environment. This involved the utilization of a private application/network server (Chirpstack framework) hosted on the cloud, as well as devices situated at the edge and extreme edge, such as PyCom devices and Lorix One.

A. Alpha and Beta Experiments

The Alpha Experiment and Beta Experiment were two separate experiments aimed at evaluating the effectiveness of a proposed approach in mitigating replay attacks. The Alpha Experiment used a LoRaWAN network without the suggested key rolling method, while the Beta Experiment incorporated the method. During the "Without Attacker" no malicious activity was performed while during the "With Attacker" phase, an attacker executed a replay attack by flooding the LoRaWAN network with packets every 4 seconds. The experiment aimed to compare the outcomes of both experiments to determine the effectiveness of the proposed solution in mitigating replay attacks.

B. Delay Experiment

The Delay experiment aimed to evaluate the key rolling mechanism's performance and the delay induced in sequential procedures. The delay refers to the duration required for retrieving an uplink message, generating a new AppKey, distributing it to the extreme edge device, and modifying the AppKey on the Chirpstack application server upon receiving an acknowledge (ACK) message.

C. Variables

For all three experiments conducted, it was assumed that the attacker had already acquired the AppKey from the "victim" device. This could have been achieved through a physical attack on the hardware of the extreme edge device or by eavesdropping on the communication link and decrypting the MIC value of an unencrypted Join Request. In order to execute these attacks, it was necessary for the attacker to be in close proximity to the extreme edge device. In addition, in all experiments, the "Frame-counter validation" was disabled to simplify the attacking method. The Frame Counter (FCNT), similar to DevNonce mentioned in the Introduction Section, is

[10] https://lora-alliance.org/resource_hub/lorawan-specification-v1-0-2

[11] https://lora-alliance.org/resource_hub/lorawan-specification-v1-0-2

an incremental number used to ensure message integrity; however, its calculation can be a time-consuming task. Disabling this process does not compromise the fundamental principles of the proposed technique.

1) Independent Variables

The three experiments had the following similar independent variables: A time window of 20 minutes for the duration, a selection that was based on the need for precise and optimal measurements, ensuring that the results obtained were both distinguishable and not excessive; a 60 sec duration of client transmission to effectively distinguish between the various packets that are received.

Additional independent variables were defined for the Alpha and Beta experiments and the Delay experiment, respectively. For the Alpha and Beta experiments: Packet data send from the extreme edge benign device: "Temp: 28 C"; Attacker device packet data: "Temp: 50.0 C". While in the Delay experiment: Packet data send from extreme edge benign device: "Temp: 32.0 C".

2) Depedent Variables

The Alpha Experiment analyzed the impact of an attack on a LoRaWAN network without a key rolling mechanism, while the Beta Experiment evaluated the impact of an attack when the key rolling mechanism was enabled. The dependent variable in both experiments was the quantity of packets received by the cloud infrastructure and stored on storage servers. On the other hand, the Delay Experiment measured the performance of the key rolling mechanism and the time window for key change delay, which are crucial in assessing vulnerabilities and potential replay attacks on LoRaWAN networks. Both experiments highlight the importance of integrating key rolling mechanisms in LoRaWAN networks.

D. Results

1) Alpha and Beta Experiments

In the analysis of the Alpha Experiment, Figure \ref{ppm} presents the findings of the Average Packets per minute (ppm) measurement. During the "With Attacker" phase, when a replay attack was executed, the ppm value was approximately 13.42. In contrast, the "Without Attacker" phase recorded a ppm value of around 1.05.

In the Beta Experiment, as depicted in Fig 8, the findings revealed that during the "With Attacker" phase, the transmission of data from the attacker to the Chirpstack application server was effectively thwarted. In the event that the packet was intercepted and an attack was carried out, the AppKey underwent modification before the attacker's packets were received. Consequently, upon arrival, these packets were promptly discarded, accompanied by an error indicating an invalid MIC value. In this experiment, the recorded average packets per minute were 4.19 ppm during the "Without Attacker" phase, and 4.09 ppm during the "With Attacker" phase.

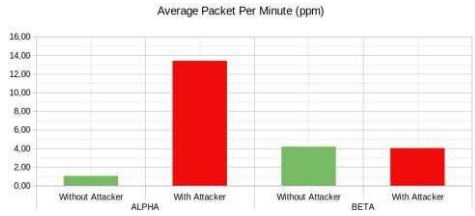

Fig 10. Packets per minute (ppm)

During the Beta Experiment, the application server effectively prevented the storage server from being flooded with duplicate or tampered data by rejecting packets with invalid MIC values. This was due to the use of a deprecated AppKey, causing packets with a MIC value associated with the deprecated AppKey to be rejected. The graph in Fig 9 shows the quantity of "Error" packets, collected from both the Beta Experiment and the Alpha Experiment. In the network, 282 packets were transmitted, with none being rejected in the Alpha Experiment. All packets, whether benign or malicious, were stored in the application server. On the contrary, during the Beta Experiment, 172 out of 257 transmitted packets were deemed unfit for further processing. Therefore, the key rolling mechanism successfully thwarted malicious actors from introducing malicious information or flooding the storage of the application server.

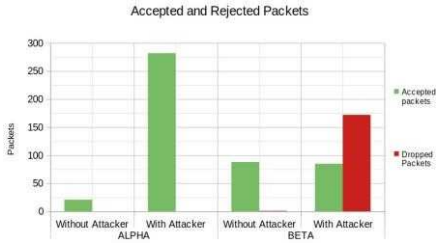

Fig 8. Accepted and Rejected Packets

2) Delay Experiment

The findings from the Delay Experiment indicate that the integration of the key rolling mechanism does not introduce any additional delay during attempts to compromise the LoRaWAN network, as it can be seen in Fig 10.

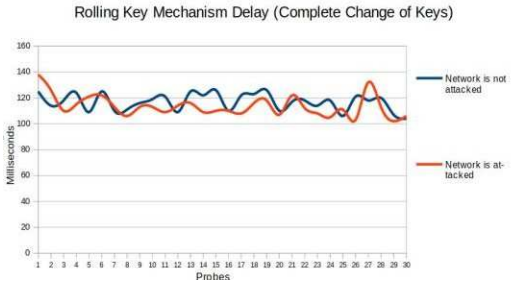

Fig 9. Delay Experiment Results

In addition, this experiment focused on quantifying the duration required for the replacement of the AppKey within the LoRaWAN network. This critical period is the only time when the network is vulnerable to potential threats, such as replay and impersonation attacks, within the proposed security solution. The vulnerability's duration was measured in milliseconds. Considering the narrow time window of 113-116 milliseconds, the chances of an attacker executing a time-consuming replay attack on the LoRaWAN network within this timeframe are extremely low. In order to carry out this process effectively, the attacker's device must make use of significant computational resources in order to intercept, decrypt, and then transmit a packet through the air to the gateway, all before the AppKey is modified again.

E. Discussion

The Alpha and Beta Experiments, along with the Delay Experiment, have shown the importance of a key rolling mechanism in enhancing LoRaWAN networks against replay attacks. The Alpha Experiment revealed the vulnerability of a

network without the proposed security solution by demonstrating the significant rise in network traffic during a replay attack. The Beta Experiment demonstrated the effectiveness of the key rolling mechanism, as it quickly alters the AppKey before every message transmission, rendering intercepted packets meaningless to attackers. The rejection of packets with invalid MIC values confirmed the mechanism's ability to distinguish between legitimate and malicious packets, increasing data and network security reliability. The Delay Experiment found minimal delays, making it nearly impossible for attackers to exploit the system within time constraints.

These findings underscore the critical role of the key rolling mechanism in safeguarding LoRaWAN networks and providing secure deployment of IoT applications. Moreover, the proposed mechanism adds an extra layer of security, even in case the application server has disabled the "Frame-counter validation" mechanism. Future research should focus on scalability, potential weaknesses in network environments, and compatibility with emerging LoRaWAN protocols. Real-world deployment scenarios and long-term evaluations will be crucial for proving the solution's efficacy.

VII. Conclusion

This paper proposed a security mechanism for LoRaWAN networks using a key rolling technique, similar to the automotive industry's security for central locking systems. The mechanism enhances the security of the Join Procedure and LoRaWAN extreme edge devices in OTAA mode by mitigating issues related to the Join Procedure in LoRaWAN networks, such as the latest 1.1. version. Although the transmission of the Join Request message over the air remains unencrypted in the proposed solution, the replacement of the static AppKey with a dynamic key for device authentication in the cloud, which changes for each message exchange session, thwarts malicious actors from executing replay attacks. As evaluation results indicated, the short time required for rolling the new AppKey prevents malicious actors from executing replay attacks as it does not allow them to eavesdrop, decrypt, and re-transmit messages with altered content or perform a replay attack before the AppKey changes again. Moreover, the proposed mechanism retains backwards compatibility while limiting complexity since only two extra packets are added in every message transfer.

However, the proposed solution does not come without limitations. For instance, the proposed solution cannot protect LoRaWAN networks from physical attacks. Additionally, the progress and use of quantum computing in the future may provide an adversary with great computational power and, thus, reduce the time required to reverse engineer the AES-128 encrypted AppKey before it changes again, in case of interception of the Join Request on a valid edge. Future implementations may consider integrating blockchain technology to enhance two-factor client authentication and real-time localization methods like RSSI to ensure data integrity. It is worth noting that the LoRa Alliance could also benefit from further exploration of this security mechanism.

References

[1] S. Ugwuanyi, G. Paul, and J. Irvine, "Survey of IoT for Developing Countries: Performance Analysis of LoRaWAN and Cellular NB-IoT Networks," Electron., vol. 10, no. 18, 2021, doi: 10.3390/electronics10182224.

[2] S. Na, D. Hwang, W. Shin, and K.-H. Kim, "Scenario and countermeasure for replay attack using join request messages in LoRaWAN," 2017 Int. Conf. Inf. Netw., pp. 718–720, 2017, doi: 10.1109/ICOIN.2017.7899580.

[3] J. P. S. Sundaram, W. Du, and Z. Zhao, "A Survey on LoRa Networking: Research Problems, Current Solutions, and Open Issues," IEEE Commun. Surv. Tutorials, vol. 22, no. 1, pp. 371–388, 2020, doi: 10.1109/COMST.2019.2949598.

[4] H. Isa, I. Bahari, H. Sufian, and M. R. Z'aba, "AES: Current security and efficiency analysis of its alternatives," 2011 7th Int. Conf. Inf. Assur. Secur., pp. 267–274, 2011, doi: 10.1109/ISIAS.2011.6122831.

[5] S. Kamkar, "Drive it like you Hacked it: New Attacks and Tools to Wireles," 2015, doi: 10.5446/36433.

[6] F. D. Garcia, D. Oswald, T. Kasper, and P. Pavlides, "Lock It and Still Lose It {\textemdash}on the ({In)Security} of Automotive Remote Keyless Entry Systems," 25th USENIX Secur. Symp. (USENIX Secur. 16), Aug. 2016, [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/garcia

[7] D.-W. Huang, W. Liu, and J. Bi, "Data tampering attacks diagnosis in dynamic wireless sensor networks," Comput. Commun., vol. 172, pp. 84–92, 2021, doi: 10.1016/j.comcom.2021.03.007.

[8] L. Migacz, X. Feng, and M. Conrad, "Automotive Security and Theft Prevention Systems: State of The Art," 2021 IEEE Intl Conf Dependable, Auton. Secur. Comput. Intl Conf Pervasive Intell. Comput. Intl Conf Cloud Big Data Comput. Intl Conf Cyber Sci. Technol. Congr., pp. 806–812, 2021, doi: 10.1109/DASC-PICom-CBDCom-CyberSciTech52372.2021.00134.

[9] A. Ghanem and R. AlTawy, "Garage Door Openers: A Rolling Code Protocol Case Study," 2022 19th Annu. Int. Conf. Privacy, Secur. Trust, pp. 1–6, 2022, doi: 10.1109/PST55820.2022.9851991.

[10] K. Hofer-Schmitz, "A Formal Analysis of EnOcean's Teach-in and Authentication," Proc. 16th Int. Conf. Availability, Reliab. Secur., 2021, doi: 10.1145/3465481.3470097.

[11] J. Kim and J. Song, "A Simple and Efficient Replay Attack Prevention Scheme for LoRaWAN," Proc. 2017 7th Int. Conf. Commun. Netw. Secur. - ICCNS 2017, pp. 32–36, 2017, doi: 10.1145/3163058.3163064.

[12] J. Thomas, S. Cherian, S. Chandran, and V. Pavithran, "Man in the Middle Attack Mitigation in LoRaWAN," 2020 Int. Conf. Inven. Comput. Technol., pp. 353–358, 2020, doi: 10.1109/ICICT48043.2020.9112391.

[13] K. Ntshabele, B. Isong, N. Gasela, and A. M. Abu-Mahfouz, "A Trusted Security Key Management Server in LoRaWAN: Modelling and Analysis," J. Sens. Actuator Networks, vol. 11, no. 3, 2022, doi: 10.3390/jsan11030052.

[14] N. Hayati, S. Windarta, M. Suryanegara, B. Pranggono, and K. Ramli, "A Novel Session Key Update Scheme for LoRaWAN," IEEE Access, vol. 10, pp. 89696–89713, 2022, doi: 10.1109/ACCESS.2022.3200397.

[15] W.-J. Sung, H.-G. Ahn, J.-B. Kim, and S.-G. Choi, "Protecting end-device from replay attack on LoRaWAN," 2018 20th Int. Conf. Adv. Commun. Technol., vol. 2018-Febr, pp. 167–171, 2018, doi: 10.23919/ICACT.2018.8323684.

[16] Z. Xia, H. Zhou, K. Gu, B. Yin, Y. Zeng, and M. Xu, "Secure Session Key Management Scheme for Meter-Reading System Based on LoRa Technology," IEEE Access, vol. 6, pp. 75015–75024, 2018, doi: 10.1109/ACCESS.2018.2883657.

[17] G. Pathak, J. Gutierrez, A. Ghobakhlou, and S. U. Rehman, "LPWAN Key Exchange: A Centralised Lightweight Approach," Sensors, vol. 22, no. 13, 2022, doi: 10.3390/s22135065.

[18] N. Hayati, K. Ramli, S. Windarta, and M. Suryanegara, "A Novel Secure Root Key Updating Scheme for LoRaWANs Based on CTR AES DRBG 128," IEEE Access, vol. 10, pp. 18807–18819, 2022, doi: 10.1109/ACCESS.2022.3150281.

[19] J. Lin, Z. Shen, M. Chen, and S. Liu, "Using blockchain to build trusted LoRaWAN sharing server," Int. J. crowd Sci., vol. 1, no. 3, pp. 270–280, 2017, doi: 10.1108/ijcs-08-2017-0010.

[20] 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. 2019 - Proc., 2019, doi: 10.1109/ICCW.2019.8756673.