IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 4 July 2022, accepted 24 August 2022, date of publication 5 September 2022, date of current version 19 September 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3204005

RESEARCH ARTICLE

Securing End-Node to Gateway Communication in LoRaWAN With a Lightweight Security Protocol

JHONATTAN J. BARRIGA AND SANG GUUN YOO, (Senior Member, IEEE)
Departamento de Informática y Ciencias de la Computación, Facultad de Ingeniería de Sistemas, Escuela Politécnica Nacional, Quito 170525, Ecuador
Smart Lab, Escuela Politécnica Nacional, Quito 170525, Ecuador

Corresponding author: Sang Guun Yoo (sang.yoo@epn.edu.ec)

ABSTRACT The arrival of IoT has brought constant innovation. This innovation has allowed many ''things'' (sensors, wearable devices, smart appliances, among others) to be connected to the Internet to deliver the information they collect. This need for connection has set the tone for the development of new protocols that adapt to the IoT environment, taking into consideration low energy consumption and low computational cost. These protocols are known as Low Power Wide Area Network (LPWAN). In this context, one of the most used is LoRaWAN. As many other IoT protocols, it is exposed to security threats. These threats aim to compromise security principles like confidentiality, integrity and availability (CIA Triad) of the information. This paper aims to analyze weaknesses related to gateways within LoRaWAN infrastructure to propose a lightweight security protocol to address gateway authentication vulnerabilities. This protocol uses lightweight cryptographic functions to achieve this goal as it is intended to be deployed over IoT devices which are very limited in terms of hardware and power resources. Likewise, this protocol has gone through a formal security analysis with the use of BAN-Logic and a tool called Scyther, to validate the security of the proposed protocol.

INDEX TERMS LoRaWAN, gateway authentication, lightweight protocol, Scyther-tool, BANLogic.

I. INTRODUCTION

The world of Internet of Things (IoT) is growing exponentially. By 2025, there will be nearly 77 billion devices connected to the Internet [1], [2], [3], [4], [5]. This growth has leveraged the emerge of certain smart initiatives (i.e. Smart Cities, Smart Campus, Smart Metering, Smart Factory, among others). Most of IoT solutions do not require high-speed connectivity, but they demand long life batteries. The amount of information transmitted by an IoT device is normally very small [6]. For instance, a temperature sensing device, would only use 7 bytes at most to inform about the temperature of a particular place.

With the aforementioned constraints (transmission and power), it is required to use lightweight secure wireless communication protocols to interconnect and exchange data among end-nodes/sensors [7], [8], [9]. These protocols or technologies are called Low Power Wide Area Network (LPWAN).LPWAN protocols support long range communications (kilometers); they are optimized for power consumption and they are not expensive to be implemented [9]. LPWAN is able to reach up to 10-40 km in rural areas whilst 1-5 km in urban areas, providing long-range communication. Moreover, a battery used in IoT devices with LPWAN has an estimate lifetime of around 10 years. Finally, IoT devices based on LPWAN are very affordable as their cost is no more than $5 dollars in some cases [8].

In terms of LPWAN protocols, there are multiple options such as LTE-M, SigFox, NB-IoT, Long Range Wide Area Network (LoRaWAN), Weightless-N, and EC-GSM. Among them, the most used are Long Range Wide Area Network (LoRaWAN), SigFox and Narrow Band – Internet of Things (NB-IoT) [7], [8], [9], [10]. These protocols have similarities in terms of architecture but differ in other parameters such as frequency of operation, security, connection fees, among others.

In comparison with other LPWAN protocols, LoRaWAN possess some benefits. First of all, its level of openness allows

The associate editor coordinating the review of this manuscript and approving it for publication was Eyuphan Bulut.

researchers to perform changes and customizations in all of its components. Moreover, there is no interconnection fee to be paid for the use of the spectrum. Finally, LoRaWAN does not require a third-party infrastructure (back-end servers) for its deployment. Anyone is free to deploy a private network by using open source tools. For these reasons LoRaWAN has been selected as the subject of study in this work.

This work reviewed papers that have identified potential vulnerabilities issues in the current version of LoRaWAN v1.1. Based on such review, this work focused on addressing vulnerabilities that have not been mitigated yet. The proposed solutions use lightweight cryptographic functions to design new protocols to mitigate potential vulnerabilities that affect the communication between end-node and gateway. Compared to other works, this approach tries to solve security problems that are not described in the specification regarding the gateway. The main contribution of this work is to propose a protocol design that could be easily implemented over any IoT LoRaWAN End-Node.

This paper is structured as follows: Section 2 provides a technical overview of LoRaWAN 1.1. and describes potential security issues. Section 3 describes improvements to be made to address some of the vulnerabilities. Section 4 analyzes the protocol from the security perspective in a formal way and performance perspective from the number of cryptographic operations. Section 5 discusses potential strengths of the current solution. Finally, Section 6 concludes the work and analyzes future improvements.

II. LoRaWAN 1.1

A. TECHNICAL OVERVIEW

Long Range Wide Area Network (LoRaWAN) is an IoT protocol that uses CSS and FSK modulation. The coverage range of this protocol oscillates between 5km within urban areas and 20 km for rural areas. It operates over unlicensed ISM bands (868 MHz in Europe, 915 MHz in North America, and 433 MHz in Asia). It has an unlimited number of messages to be sent per day. In terms of bandwidth it supports 125Khz and 250Khz. A payload can handle up to 243 bytes. It implements mutual authentication with the use of two symmetric keys. For encryption it uses AES-128 in CTR mode and for integrity it uses Message Authentication Codes (MAC). Its infrastructure is completely open and allows private implementations given the chance that anyone could implement his own infrastructure by using open source tools like ChirpStack (https://www.chirpstack.io/) [3], [7], [8], [9], [11].

LoRaWAN operates at the MAC layer and it is based on LoRa (physical layer protocol). LoRa is the physical layer protocol. It is based con Chirp Spread Spectrum which is similar to FSK modulation, but it provides a longer communication range [12].

LoRaWAN has gone through several improvements so that its specification has changed several times. The specification which will be considered for this work will be [13] which has major changes in terms of session keys.

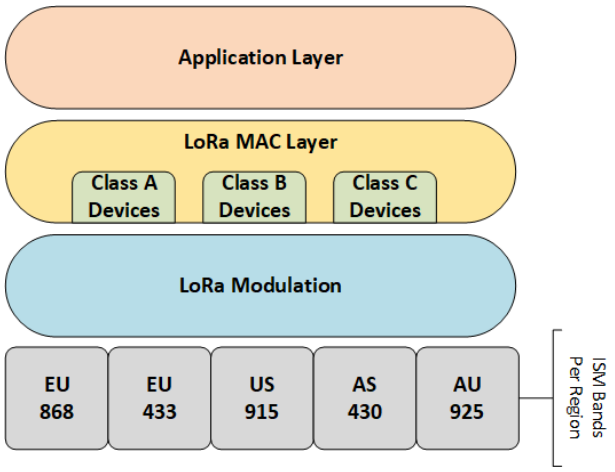

FIGURE 1. LoRaWAN classes.

LoRaWAN operates over unlicensed Regional Industrial Scientific Medical (ISM) bands. ISM bands are 868 MHz in Europe, 915 MHz in North America, and 433 MHz in Asia. LoRaWAN has three classes known as Class A, B and C as shown in Figure 1. Class A is not optional and has to be implemented by all end-nodes. Devices that implement more than the mandatory class are considered High-End devices [13].

There are three classes of devices according to LoRaWAN specification. First, class A devices are bi-directional end nodes which are more energy efficient and have two short defined reception windows after every uplink message. Class B devices open additional receive windows on scheduled times with the use of beacons sent by the gateway. On the other hand, Class C devices are continuously listening and they are the least energy efficient but offer the lowest latency level [13].

1) ARCHITECTURE

LoRaWAN is composed of three elements: end-nodes, gateways and back-end servers. On the other hand, back-end servers are composed of: Network Server *(NS)*, Join Server *(JS)* and Application Server *(AS)*. Any end-node that wants to communicate with the back-end server infrastructure must go through a gateway (*Gw*). The communication between end-node (*EN*) and gateway is performed through LoRa protocol which is based on Chirp Spread Spectrum [12]. *Gw* to back-end servers communication is handled over TCP/IP protocols [9], [10], [12]. The following Figure 2 shows the architecture of LoRaWAN with all its actors.

2) LoRaWAN BACKEND INFRAESTRUCTURE

As described in LoRaWAN Backend Interfaces Specification [14], besides radio gateway, there are three types of servers that are part of the backend architecture of LoRaWAN. Those servers are: Network Server, Application Server *(AS)*, and Join Server; each of them perform specific tasks within the whole architecture. The Network Server *(NS)* is in charge of handling LoRAWAN MAC layer for end-nodes, forwarding messages to *AS*, forwarding Join

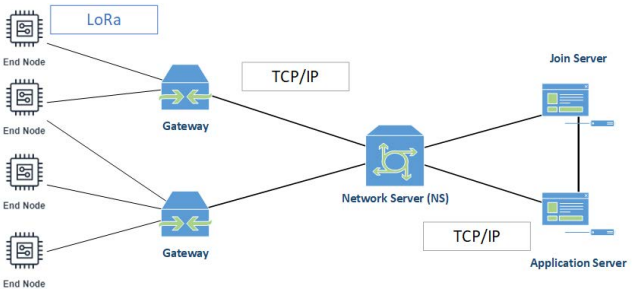

FIGURE 2. LoRaWAN architecture.

messages to *JS*, frame authentication, end-node verification among others. For Roaming scenario, LoRaWAN Backend Interfaces Specification [14] describes three roles for the NS which are home *(hNS)*, *serving(sNS)* and *forwarding(fNS)*. *hNS* is responsible for persisting information related to Service, Device and Routing profile, and *DeviceEUI*. These roles depends on *JS* for joining purposes and is connected to *AS*. In roaming scenario *sNS* and *hNS* are separated and uplink or downlink messages are passed from *sNS* to *hNS*. *hNS* is in charge of forwarding uplink messages to the proper *AS* based on *DevEUI* parameter. In addition, *sNS* role handles only MAC layer for the End-Node. Last, *fNS* handles gateways and there may exist more than one *fNS* serving a single End-Node. According to [14] *JS* manages End-Device activation process through Over the Air Activation (OTAA). A single *JS* could be connected to multiple *NSs*. This server contains information concerning Join-Request frames (uplink) and Join-Accept frames (downlink). It shares derived sessions keys with *AS* and *NS*. A *JS* could be connected to several *AS*, also a single *AS* could be connected to multiple *JS*. The *AS* is in charge of handling payloads (uplink and downlink frames) sent by the End-Devices. *AS* may be connected to multiple *NSs* and *JSs*, and several *AS* may be connected to a single *NS*. According to [14], there are several interfaces in place to support several procedures within LoRaWAN network from the perspective of the End-Device (home or roaming). These interfaces are:

- ***sNS - JS***: Used during Roaming Activation Procedure, it helps to obtain *NetID* from *hNS* of a particular *EN*.
- ***hNS - JS***: Supports Join Procedure between *NS* and *JS*.
- ***hNS - sNS***: Supports signaling whilst in roaming as well as payload delivery between *hNS and sNS*.
- ***sNS - fNS***: Supports signaling whilst in roaming as well as payload delivery between *sNS* and *fNS*.
- ***AS - hNS***: Supports payload delivery between *AS* and *hNS*.
- ***AS - JS***: Supports delivery of Application Session Key *(AppSKey)*.
- ***EN - NS***: Used to support LoRaWAN MAC-layer signalink and payload delivery between *EN* and *NS*.

The procedure for activating and End-Node (EN) within the LoRaWAN infraestructure is known as *Join Procedure* or *Over The Air Activation (OTAA)*. This procedure is described in detail in the next section.

3) JOIN-PROCEDURE AT HOME SCENARIO

LoRaWAN supports two activation processes (join procedures) for enabling end-nodes over a LoRaWAN network. Those processes are: Activation By Personalization (ABP) and Over the Air Activation (OTAA).

The OTAA procedure is started by the end-node. For this purpose, each end-node has the following security parameters $DeviceEUI_{ENi}$, $JoinEUI_{ENi}$, $NwkKey_{EN_i}$ *and* $AppKey_{EN_i}$. The last two parameters are 128-bit keys used to derive session keys. These parameters are factory stored settings. This procedure is considered more secure than ABP since other keys are derived from known parameters stored in the device. In ABP mode, it is required that all sessions keys have to be preloaded in the end-node, application server, network server and join server for executing the join procedure and then sending uplink messages.

The Join-Procedure is a process for authenticating *End-Nodes* over a LoRaWAN network. This process is mandatory before sending any uplink message. In order to proceed the End-Node must first build a Join-Request message composed as follows by the *JoinEUI*, *DevEUI* and the *DevNonce*. *JoinEUI* is an identifier of the *JS, DevEUI* is a unique identifier of the Device and *DevNonce* is a sequential 2-byte number generated by the *EN*. This message is sent in plaintext. These parameters are evaluated by the *JS* and *NS* as follows. *JS* verifies that *DevEUI* is in the authorized list whilst *NS* validates and keeps a track of every *DevNonce* generated. If the procedure is successful, the Network Server will respond with a Join-Accept message to the *EN* so that it could derive session keys (Application Session, Network Session and Join Session keys) [10], [13].

The Join-Accept message contains the following parameters *JoinNonce, Home_NetID, DevAddr DLSettings, RxDelay and CFList* [13]. The following table describes the parameters that are part of the Join-Accept message see Table 1.

Once the *EN* receives the Join-Accept message, the following session keys are derived according to the specification [13]. Every session key is used for a particular purpose. ***FNwkSIntKey*** and ***SNwkSIntKey*** are used to calculate MIC fields for preserving message integrity.***NwkSEncKey*** is used to cypher messages for NS.***AppSKey*** is used to cypher Frm-Payload for ***AS*** [13]. The session keys are derived as follow according to the specification:

FNwkSIntKey$=SEnc(NwkKey,0x01||JoinNonce||JoinEUI||DevNonce||pad16)$

SNwkSIntKey$=SEnc(NwkKey,0x03||JoinNonce||JoinEUI||DevNonce||pad16)$

NwkSEncKey$=SEnc(NwkKey,0x04||JoinNonce||JoinEUI||DevNonce||pad16)$

AppSKey$=SEnc(AppKey_{EN_i},0x02||JoinNonce||JoinEUI||DevNonce||pad16)$

JSIntKey$=SEnc(NwkKey_{EN_i},0x06||DevEUI_{ENi}||pad16)$

After the Join-Accept, *JS* must record and keeps a track of every JoinNonce generated every time a Join or a Rejoin is performed.

TABLE 1. Join-accept parameter summary.

Parameter	Size (bytes)	Generator	Purpose
JoinNonce	*3*	*JS*	Counter value incremented with every Join-Accept
NetID	*3*	*NS*	Network Identifier
DevAddr	*4*	*NS*	Device Address
DLSettings	*1*	*NS*	Downlink Configuration
RxDelay	*1*	*NS*	Delay between reception and transmission
CFList	*16*	*NS*	Optional list of network parameters

On the other hand, ABP procedure requires the manual input of session keys listed before. This procedure does use the same session keys for all their lifetime. It is then, more insecure than OTAA. There is no join-procedure or session key derivation and if keys are required to be renewed, they need to be manually configured.

Although there are some security considerations and activation processes described in the specification, there are still some issues that need to be addressed as they may compromise integrity and confidentiality of data and actors. These issues are described next.

B. SECURITY ISSUES IN LoRaWAN

LoRaWAN like any other LPWAN protocol, takes into consideration the typical limitations of constrained IoT devices. These constraints limit the ability to provide higher levels of integrity, confidentiality, and availability. However, LoRaWAN itself is not computationally constrained as payloads might reach 242 bytes over US915 frequency considering a Spread Factor (SF) of 7 a bandwidth of 125kHz [15].

This protocol has gone under some improvements, particularly in the security field with the specification released in 2017. This specification is currently under version 1.1 and is maintained by LoRa alliance [13]. This version corrects several vulnerabilities identified previously [16], and adds important features like the inclusion of a second key and separation of duties. Nevertheless, the improvements made to the specification have not addressed several issues like bit-flipping attacks, channel eavesdropping, rogue gateway attacks among others as described in [10], [17], and [18]. These attacks represent a threat for applications developed under this technology. LoRaWAN is a popular protocol and it is being deployed in several applications in different areas such as health care, smart city, smart farming, environmental monitoring, geolocation among others. However, it still presents vulnerabilities that can affect the deployment of solutions as well as end-users.

In terms of research, several works have been published to increase the level of security in LoRaWAN. Most of the reviewed papers are oriented to correct weaknesses over LoRaWAN v1.0. Published works are more oriented to deal with secure key distritbution and generation in order to secure keys that are the root component to guarantee confindentiality and privacy of the payloads generated. Very few papers analyze security vulnerabilities in LoRaWAN 1.1 version and only a few propose some improvements. Table 2 shows some of the reviewed works, their research focus and the version of LoRaWAN used for the research [10].

TABLE 2. LoRaWAN research summary in terms of vulnerabilities.

Reference	Research Focus	LoRaWAN Version
[10]	Gateway Attacks	1.1
[19]	Gateway Attacks	1.0 and 1.1
[17]	Security Analysis	1.1
[16], [20], [21]	Gateway Attacks	1.0
[22]	Security analysis	1.0
[23]–[27]	Key management improvements	1.0
[28]	Key management distribution improvements with blockchains	1.1
[29]	Root key protection in JS	1.1
[30]	Formal security verification	1.0 and 1.1
[31]	Join Procedure backward compatibility	1.0
[32]	Replay attacks	1.0
[33]	Replay attacks	1.0
[34]	Join Procedure with blockchains	1.1
[35]	Jamming attacks	1.0

As can be seen in the previous table, there is very few research being carried out in terms of vulnerabilities associated to gateways. In spite of recommendations that have been made, there are no formal requirements in the specification that focus on mitigating gateway vulnerabilities. In this situation, this research will be oriented on securing the communication between the *Gw* and the backend infrastructure.

For its part, the work presented by [20] discusses about impersonating gateways. Although the author performs this analysis over LoRaWAN 1.0, they are still aplicable as the current specification does not address any security requirements or improvements to the gateway. The author describes that registering a gateway is not a mandatory requirement. The proof of concept that he perfomed used the platform The Things Network (TTN). This platform is able to accept traffic from unregistered gateways and they mark traffic as unstrusted to differentiate from traffic generated by registered gateways. This attack takes place in four stages as described by the author. First, a malicious user has to acquire the gateway unique id. Then, the gateway gets disabled by the attacker. Once the id has been obtained, the malicious gateway is configured with the original id to communicate with the valid Network Server. Finally, a malicious user is able to perform an ACK spoofing. This attack is discussed and tested by [16] where they showed that a gatewway can selectively decide which packets not being transmitted. Under the described scenario, the attacker would have physical access to the device or perform a jamming attack to completely disable it. As the used platform does not perform any further validations, once the malicious gateway ''assumes'' the identity of the compromise gateway it will be able to push messages through a malicious device. Finally, the author propose some countermesaures to prevent this attack from happening. The author suggests using IDS devices to detect a change of the IP address, and putting gateways in a safe and secure place. Moreover, the authors indicate that having gateway

redundancy is an alternative. Securing the channel between gateway and the backend infrastructure is also a suggested countermeasure. Last but no least, dereasing spread factor would notably reduce the possibility of a jamming attack.

On the other side, the authors in [19] perform a review on vulnerabilities over gateways and propose possible solutions. That work reviews problems like: Radio Jamming Attack, Beacon Attacks, Eavesdropping, Replay Attack, Wormhole attack and Rogue Gateway Attacks. i) Radio Jaming Attacks target the physical layer of LoRaWAN because end-nodes use radio signals to communicate with gateways. These signals can be interrupted through malicious hardware. Authors describe triggered and selective attacks. Triggered ones are easier to detect as they target all devices whilst selective affect a specific device only. ii) Beacon Attacks according to authors may affect class B devices since those beacons are not secured, a fake gateway might cause packet collisions leading to a Denial of Service (DoS). iii) Eavesdropping mainly affects packets that may be encrypted with the same key. iv) Replay attacks as mentioned by authors is an efficient attack against gateways as an attacker is able to send packets as if it were authorized. An attacker uses the highest value of the counter to repeat messages. v) Wormhole attacks consist on capturing packets and retransmitting it to another location in the network. This attack might cause downling messages to be routed to an invalid location. Although there are integrity checks within the packet sent, the receiver server does not validate the packet or its place of origin. vi) Rogue Gateway Attacks are possible since gateways are not authenticated in any way within the protocol.

The authors in [10] defines the *GW* as the weakest link of the communication between *EN* and the backend infrastructure. They emphasize that at this level, it is posible to perform any kind of capture or physical attack. In addition, authors state that gateways has been catalogued as ''reliable'' from the very beginning of LoRaWAN. In their work, they describe that there are two potential attacks that can take place. First, Beacon synchronization DoS attacks produced by attackers using untrusted *GWs* to send fake packets to affect synchronization of Class B devices. Moreover, *GWs* can be impersonated to sniff traffic generated by an authorized *EN* and therfore determine their network address. Also, the physical location of a *EN* can be determined by using a triangulation method. Finally, authors suggest that a mechanism to address this issue would be to implement mutual authentication between *EN-GW* and *GW-NS*.

A brief overview of vulnerabilities over some IoT protocols is reviewed in [21]. The authors in their work aim to brute-force the MIC of a LoRaWAN packet. In their test they aim to build forged packets by calculating the MIC of future packets based on captured data. Although the are not fully able to forge packets, they made the server to accepted a packet with incorrect data. The authors highlight that this scenario is possible due to an insecure communication between *Gw* and *NS* leading to a Man-In-The-Middle-Attack (MiTM). Also, the authors highlight that this scenario is feasible to exploit as the Semtech Packet Forwarder does not guarante an authenticated connection between *Gw* and *NS*.

TABLE 3. Scyther results claims with Gw.

Role	Claim	Status	Attack patterns
End-Node	Alive	*Fail*	1 attack
	Weakagree	*Fail*	1 attack
	Niagree	*Fail*	1 attack
	Nisynch	*Fail*	1 attack
Gateway	Alive	*OK*	No attacks
	Weakagree	*Fail*	1 attack
	Niagree	*Fail*	1 attack
	Nisynch	*Fail*	1 attack
Server	Alive	*Fail*	1 attack
	Weakagree	*Fail*	1 attack
	Niagree	*Fail*	1 attack
	Nisynch	*Fail*	1 attack

C. GATEWAY ATTACKS

1) FORMAL SECURITY VERIFICATION OF LoRaWAN 1.1 WITH THE INCLUSION OF THE GATEWAY ROLE

The authors in [30] performed a formal security verification by using a tool known as Scyther to analyze the state of security of the Join-Procedure in the LoRaWAN protocol for both versions 1.0.3 and 1.1. In such work, they demonstrated that LoRaWAN v1.1 is more secure. The authors considered only the end-node and the Join Server, but they did not include the gateway as it acts as packet translator to deliver uplink messages to the back-end infrastructure; however, *Gw* plays a crucial role as if it fails packets could not be delivered.

In our approach, we will use the same tool but considering the role of the Gateway in the protocol and taking the same assumptions as described in [30]. The following results were obtained after running the Scyther tool (see Table 3).

The results show that the inclusion of this role produces an affectation in the protocol. This inclusion has affected the *Weakagree* principle, meaning that the partners might be communicating with an intruder. The other claim affected is *Niagree*, which means that the parties are not able to agree on the value of variables after execution. Besides, the claim *Nisynch* is also affected. It means that the protocol is not executed in order and that contents cannot be preserved during communication. Failing the the claim of synchronicity *Nysinch* implies that there is no mutual authentication between gateway and server (Network Server). Therefore, the gateway must be authenticated within the LoRaWAN infraestructure as this claim fails also between End-Node and Gateway. The table 3 shows that either the End-Node *(Dev)*, Gateway *(Gw),* or Join Server *(JS)* could be attacked. This attack can be replicated not only during the Join-Procedure but also during uplink and downlink messages.

Uplink messages contains information payloads generated by the end-nodes that are intended to be delivered to an application server for further data processing and analysis. This process could be executed once an end-node has been activated either through OTAA or ABP join-procedures. The uplink message delivery protocol is shown in Figure 3.

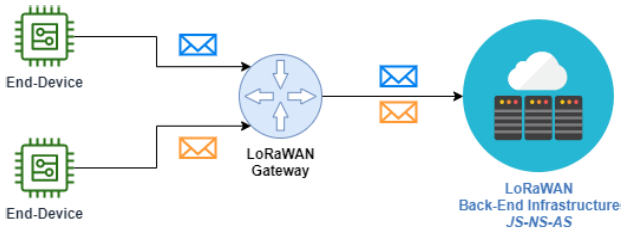

FIGURE 3. Uplink message protocol.

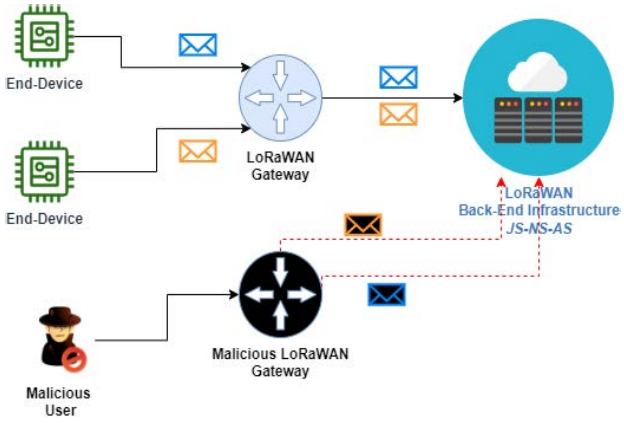

FIGURE 4. Malicious gateway injecting packets.

2) THREAT MODEL

For defining the threat model, the uplink message protocol will be considered. The following assumptions are in place. The device has already approved successfully the OTAA Join-Procedure. The communication between the *End-Node* and the *Gw* flows over an insecure open channel. The *Gw* is not an authenticated device over the infrastructure. The adversary could be either internal or external. According to the specification this issue is not specified and assumes that the gateway is a "trusted" device. This lack of authentication allows malicious users to produce the following attack scenarios:

1) A malicious (not authenticated) gateway deployed for injecting captured or fake packets to a network server of a real LoRaWAN network as shown in Figure 4, might affect network server availability due to processing an excesive number of unauthorized packets.
2) A malicious attacker deploying a rogue-gateway to sniff the traffic and perform cryptoanalysis over the packets that are delivered to a valid back-end infrastructure affecting confidentiality and integrity of information as shown in Figure 5.

From the scenarios described before, a malicious user is able to compromise the communication channel between the End-Node and the Gateway as it is not protected. Although the frame payloads are encrypted, a malicious user is able to decode PHYPayload. According to LoRaWAN specification [13] a PHYsical PAYLOAD of an uplink message is composed as shown in Figure 6.

Any user is able to decode information sent from a PHYPayload of LoRaWAN as only some parameters of it are

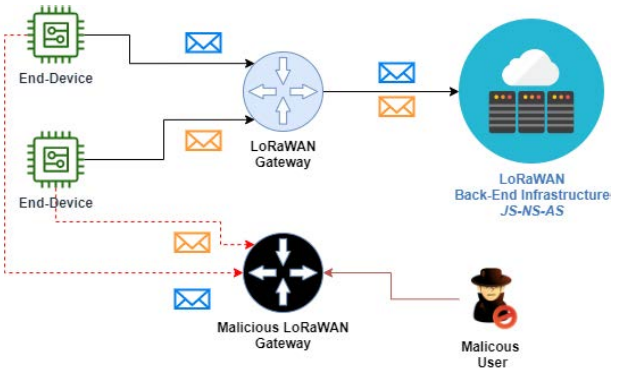

FIGURE 5. Channel eavesdropping.

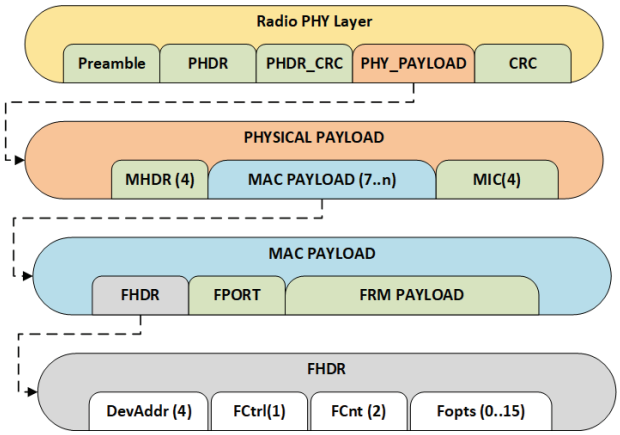

FIGURE 6. LoRaWAN PHY payload structure.

encrypted. The information that can be obtained by applying a simple Base64 or Hex decoding is listed in the following Table 4.

Although a malicious user would have a hard time trying to decrypt FRMPaylod, it is important to denote that LoRaWAN session keys live for a long a time and they are just renewed with rejoin procedure or by sending a *RekeyInd* command that is triggered by the device and shall be processed by the *NS* to produce a new pair of keys [13]. With the aforementioned before, a cryptoanalysis procedure is valid to infer the content of the FRMPaylod considering that the minimum length is 7 bytes protected with an AES-128 bit key. In addition, as a malicious user is able to capture packets he is also able to resend this packets or craft malicious packets based on previous informaiton and send them to the LoRaWAN infrastructure since the gateway is assumed to be "trusted". The exposed vulnerabilities require a mechanism that prevent malicious users to easily sniff, divert or inject unathorized traffic.

Within LoRaWAN infrastructure, gateways are a key element since they convert LoRa frames into IP packets; therefore, they could be considered as a point of failure, which may affect packet delivery to the backend infrastructure or downlink messages to the End-nodes. Because of their importance, these elements must be recognized within the network to guarantee their authenticity when forwarding or

TABLE 4. LoRaWAN parameters obtained through decoding.

Parameter	Obtained by decoding
MHDR	Yes
MACPayload	Yes
MIC	Yes
FHDR	Yes
FPort	Yes
FRMPayload	No
DevAddr	Yes
FCtrl	Yes
FCnt	Yes
FOpts	No

delivering messages. If gateways are not authenticated within LoRaWAN infrastructure, they are exposed to rogue gateway attacks which may affect their availability and may affect the normal flow of packets. An authenticated gateway will enhance the security level of messages, it would protect/cypher certain data from FHDR to prevent cryptoanalysis or using a known key to forward messages to authenticated network servers only. Likewise, this authenticated gateway would be the only authorized to send class B beacons to End-Nodes, these beacons might contain additional validation fields to prevent Class B nodes being flooded by such type of messages. The proposed approach aims to provide an enhancement over the *End Node-Gw* and *Gw-NS* relationships to use only registered gateways within LoRaWAN infrastructure by using lightweight cryptography protocols without requiring third-parties like Certification Authorities (CA).

III. SECURING END-NODE TO GATEWAY COMMUNICATION IN LoRaWAN

To achieve the proposed goal, the following solutions are proposed. First of all, register the gateway through the Network Server (*fNS* if in roaming scenario) by generating a new key to authenticate it over the LoRaWAN infrastructure. Also, produce a new session key that will be used between the *EN* and the *Gw*. This key will be known as *GwSKey* and will be generated during the Join Procedure. It will be shared to the *NS* and later to the *Gw* or group of *Gw* tied to a particular *NS*. Table 5 shows the notation used in the following protocols.

A. GATEWAY REGISTRATION PROTOCOL

This protocol registers a gateway within a LoRaWAN network. During this registration, the Gateway (*Gw*) will share its symmetric key with the Network Server (*NS*). In this scenario, it is assumed that the gateway symmetric key will reside in secure place that cannot be tampered.

In the proposed scenario each *NS* is in charge of one or a group of gateways. According to the LoRaWAN backend specificaction V1.0 [14], *fNS* is in charge of managing Gateways. A Gateway points to a particular *fNS*. There might be several *Gws* deployed within the network and connected to a network server. The number of gateways depend on the number of nodes that can be handled and the scope of the deployed network. This protocol aims to mitigate the vulnerability described as Rogue Gateway attacks. This is a formal protocol that must take place before any *Gw* wants to be part of a LoRaWAN network. The process for registering a gateway into the LoRaWAN network is executed as follows (see Figure 7). For this scenario, it is assumed that the network administrator has to configure the Gw_i to connect to a *fNS* or a set of them.

First, the user in charge of performing the configuration is a network administrator which provides his/her credentials ID_{Ui}, PW_{Ui} into the gateway. Then, the gateway Gw_i generates a randmon nonce *RN1*, a random symmetric key *RSK*, computes $\boldsymbol{GwSKa}=h(RSK \,||\, GwKey_{Gwi})$, where $GwKey_{Gwi}$ is a symmetric key that comes from factory and is stored in a secure place in the gateway, and calculates $\boldsymbol{GwInf}=SEnc(GwSka, RN1||GwEUI_{Gwi})$ where $GwEUI_{Gwi}$ is a 64-bit Unique Identifier of the gateway, and $SEnc(x,y)$ is a symmetric encryption function y using the key x, $||$ is a concatenation operation, and $h(.)$ is a one-way hash function. Then, it calculates the following:

- $\boldsymbol{MReq} = h(ID_{Ui}||h(PW_{Ui})) \otimes GwSka$
- $\boldsymbol{M1} = (MReq||IDUi||GwInf)$

The gateway (Gw_i) communicates with the *fNS* and asks for gateway registration by sending *M1*. After receiving the request, *fNS* Obtains ID_{Ui} from *M1* and calculates $h(IDUi||h(PWUi))$. It obtains $\boldsymbol{GwSKa}$ by executing $MReq \otimes h(IDUi||h(PWUi))$. It extracts $RN1||GwEUI_{Gwi}$ by performing $SDec(GwSKa, GwInf)$ where $SDec(x,y)$ is a symmetric decryption function of message y using key x. It generates two random nonces *RN2* and *RN3* and then computes the symmetric groupkey key for all gateways associated to *fNS* by executing $\boldsymbol{GrpKey_{GrpId}} = h(GwKey_{Gw_i}||GwKey_{Gw_j}||GwKey_{Gw_{j+1}}||GwKey_{Gw_{j+n}}||RN3)$, where $GwKey_{Gw_n}$ is a symmetric key that belongs to a particular registered gateway n. This key is the group symmetrickey that will be used by the *fNS* to share multicast messages with its registered gateways. Then, it generates a sequential integer *GrpId* to identify the group of gateways connected to it. It stores $(GwEUI_{Gwi}, h(GwEUI_{Gwi}), GrpKey_{GrpId}, GrpId)$ in its LocalDB. For this scenario, every time a gateway is registered, the $GrpKey_{GrpId}$ will be calculated and shared (multicast) to all the gateways tied to a *fNS*. Finally, it calculates $\boldsymbol{M2}=SEnc(GwSka, GrpKey_{GrpId}||RN1'||RN2)$ and sends it back to Gw_i.

Once Gw_i receives *M2*, it obtains $GrpKey_{GrpId}||\ RN1'||\ RN2'$ by decrypting $SDec(GwSKa, M2)$. Then, it validates if the received random nonce *RN1'* matches the previously generated one *RN1*, to guarantee the freshness of the message. If previous validation was ok, it calculates $\boldsymbol{MICGw} = aes\_cmac(GrpKey_{GrpId}, RN2'||GwEUI_{Gwi})$, where $aes\_cmac(x,y)$ is an AES Message Authentication Code function that uses a key x to produce a code of a message y. Then, it computes $MA=SEnc(GrpKey_{GrpId}, RN2'||MICGw)$ Finally, it calculates $\boldsymbol{M3} = MA||h(GwEUI_{Gw_i})$ and sends $\boldsymbol{M3}$ to *fNS*.

Finally, upon reception of $\boldsymbol{M3}$, *fNS* obtains $h(GwEUI_{Gw_i})$ and compares against its LocalDB to obtain the $\text{GrpKey}_{\text{GrpId}}$

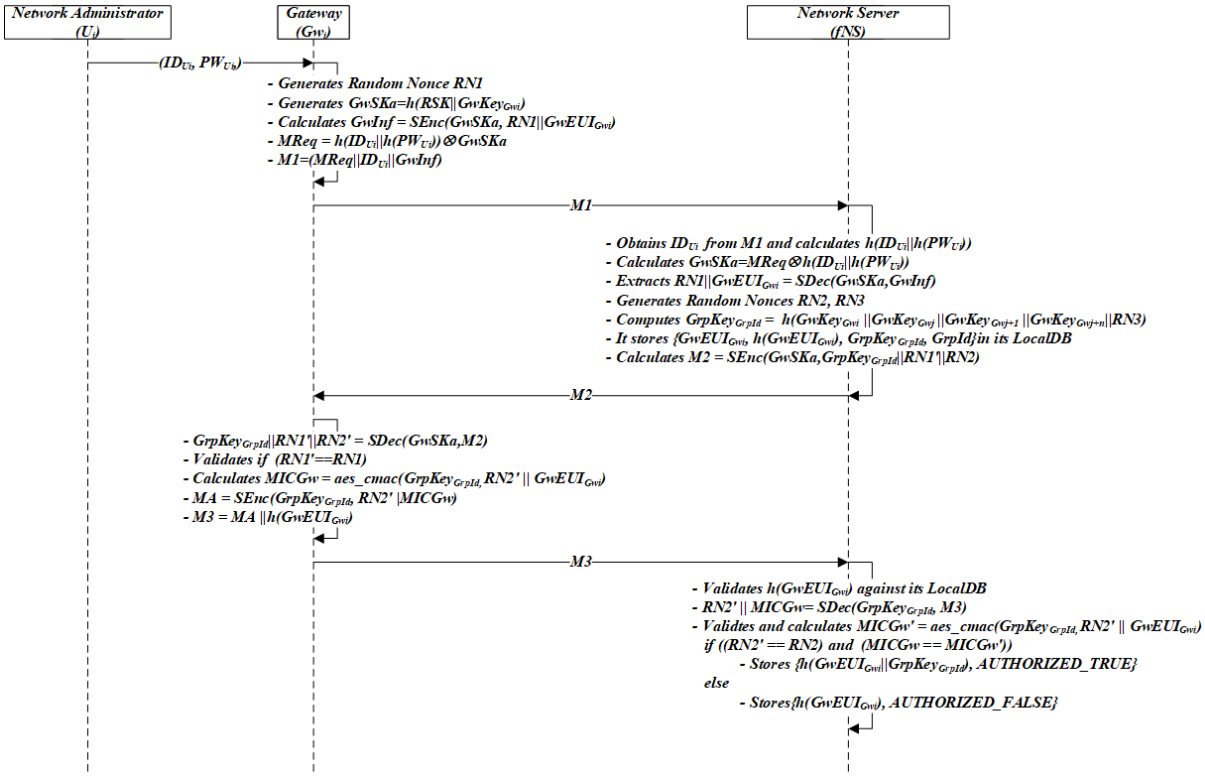

FIGURE 7. Gateway registration protocol.

of the gateway that is requesting the registration process used for further decrytption operations. It decrypts *MA* to obtain *RN2' and MICGw* by executing *SDec*($GrpKey_{GrpId}$, *MA)* using the key retrieved from its LocalDB. Then, *fNS* calculates ***MICGw'*** by executing aes_cmac($GrpKey_{GrpId}$, RN2||$GwEUI_{Gwi}$) where *RN2* is the previous random nonce generated by *fNS*. It compares ***MICGw'*** against the received ***MICGw*** to validate that the message has been generated by the gateway requesting registration. Also, it validates *RN2* against *RN2'* to validate the freshness of the message. If both comparisons are valid, *fNS* stores the tuple *(h($GwEUI_{Gwi}$||$GrpKey_{GrpId}$), AUTHORIZED_TRUE)* in its database to authorize messages coming from the just registered gateway. Otherwise, it prohibits the gateway by registering the tuple *(h($GwEUI_{Gwi}$), AUTHORIZED_FALSE)*. For future use, the *fNS* will first validate the authorization status of a gateway before accepting/forwarding packets to other network servers.

The proposed scenario applies for home or roaming scenarios. In case of home deployment, *NS* acts as *fNS, sNS* and *hNS* according to the LoRaWAN backend interfaces specification [14]. In our proposal the *fNS* plays the role of *NS*.

It is important to consider that if a gateway Gw_i leaves or enters the group, then a recalculation procedure should be conducted by *fNS* and the resulting key must be shared among the group through multicast. For the leaving scenario, the multicast operation will not consider the leaving gateway, it will unauthorize Gw_i in the *fNS* database by registering the tuple *(h($GwEUI_{Gwi}$), AUTHORIZED_FALSE)* and calcuating the new group key as follows $\boldsymbol{GrpKey_{GrpId}} = h(GwKey_{Gw_j}||GwKey_{Gw_{j+1}}||GwKey_{Gw_{j+n}})$.

B. GATEWAY SESSION KEY DERIVATION PROTOCOL

1) HOME SCENARIO

The process for deriving the Gateway Session Key ***(GwSKey$_{ENi}$)*** in a LoRaWAN Home Scenario is executed as follows (see Figure 8). The steps highlighted in green are part of our contribution.

This procedure is executed during the Join-Procedure OTAA described in the LoRaWAN 1.1 specification. Once the *EN* has passed all validation procedures by *NS* and *JS*, it starts the Session Key Derivation Process. According to the specification, there are five keys that are derived and shared with the Network Server and the Application Server. In this scenario, a new symmetric key based on previous Network Key and Application Key is calculated by using an XOR function and then using it to calculate a sixth session key known as $\boldsymbol{GwSKey_{ENi}}$. The following steps are executed:

- $\boldsymbol{NAKey_{EN_i}}$=$NwkKey_{ENi} \otimes AppKey_{ENi}$
- $\boldsymbol{GwSKey_{ENi}}$=$aes\_cmac(NAKey_{EN_i}, h(DevEUI_{ENi}||DevNonce_{ENi}||JoinNonce_{ENi}||JoinEUI_{ENi}))$

Once obtained, *JS* generates *M1* and asymmetrically encrypts it with the public key of the Netowrk Server *(NS)* NS_{Pubkey}. *JS* is able to determine $GwEUI_{Gwi}$ as it comes in the payload of the Join-Procedure. JS computes:

TABLE 5. Notations used in designed protocols.

Notation	Description
Gw_i	Gw_is' device
U_i	i th user
RN1, RN2, . . . , RNn	Random nonces
EN_i	End-node$_i$
$AppSKey_{EN_i}$	Session key used to cypher data to AS from End-node
$NwkSEncKey_{EN_i}$	Session key used to cypher data to NS from End-node
$JSIntKey_{EN_i}$	Network Session key derived during OTAA
$GwSKey_{EN_i}$	Session key used to cypher data to Gw from End-node
$SNwkSIntKey_{EN_i}$	Session key used to calculate partial MIC over uplink messages, full MIC on downlink messages and rejoin request.
$FNwkSIntKey_{EN_i}$	Session key used to calculate partial MIC over uplink messages.
$GwKey_{Gw_i}$	Gateway Symmetric key
$GwKey_{Gw_{j..n}}$	Symmetric keys of other gateways
$AppKey_{ENi}$	Pre-shared root application key
$NwkKey_{ENi}$	Pre-shared root network key
$NAKey_{EN_i}$	Calculated key between $AppKey_{ENi}$ *XOR* $NwkKey_{ENi}$ to derive $GwSKey_{EN_i}$
$GwEUI_{Gw_i}$	Gateway Extended Unique Identifier
$DevEUI_{EN_i}$	Device Extended Unique Identifier
$GwDevId_{EN_i}$	End-node anonymous identity
$GrpKey_{GrpId}$	Symmetric Group Key for multicast messages
GrpId	Group Identifier of a set of gateways connected to a *fNS*
MICGw	MIC used to validate integrity between *fNS* and Gw_i
$Pubkey_{fNS}$, $Privkey_{fNS}$	*fNS*'s asymmetric key pair
$Pubkey_{NS}$, $Privkey_{NS}$	*NS*'s asymmetric key pair
$Pubkey_{JS}$, $Privkey_{JS}$	*JS*'s asymmetric key pair
ID_{Ui}	Identification of U_i
PW_{Ui}	Password of U_i
||	String concatenation
h.	One way hash function
$\otimes$	Exclusive OR operation
mic.	Message Integrity Code function
aes_cmac	AES Message Authentication Code function
SEnc(x,y)	Symmetric encryption of message y using the key x
SDec(x,y)	Symmetric decryption of message y using the key x
MICPy	Additional MIC to protect Payload Integrity
$MIC\_P_{EN_i}$	MIC to validate message Integrity between EN_i and Gw_i
JS	Join Server
NS	Network Server
AS	Application Server
$DevAddr_{ENi_i}$	Device Address assgined by the network server
MHDR	MAC Header
FHDR	Frame Header
Mtype	Message Type
EJP	Extended For Join Protocol (RFU unused bits)
FPort	Optional Port Field
FCtrl	Frame Control
FCnt	Frame Counter
FOpts	Frame Options
Payload	Unencrypted Message Payload
FRMPayload	Encrypted Frame Payload
B0	Uplink B0 MIC computation block format
B1	Uplink B0 MIC computation block format
msg	Whole message that is composed of *MHDR, FHDR, FPort, FRMPayload*

- $\boldsymbol{M1}$=$AEnc(NS_{Pubkey}, GwEUI_{Gwi}||DevEUI_{EN_i}||GwSKey_{EN_i})$
- $\boldsymbol{GwDevId_{EN_i}}$=$h(GwSKey_{EN_i}||DevEUI_{EN_i})$

JS stores $\{GwDevId_{EN_i}, DevEUI_{EN_i}\}$ for further processing, *M1* is sent back to the network server by using the sharing process of session keys and JoinAccept is forwarded to End Node.

The network server *(NS)* receives *M1* and decrypts it by executing $ADec(NS_{Pubkey}, M1)$ to obtain $GwEUI_{Gwi}||DevEUI_{EN_i}||GwSKey_{EN_i}$, where *ADec(x,y)* is an asymmetric decryption function that uses a public key *x* to decrypt a message *y*. Then, it calculates ***M2*** by executing $\boldsymbol{M2}$=$SEnc(GrpKey^{GrpId}, GwEUI_{Gwi}||DevEUI_{ENi}||GwSKey_{ENi})$ and sends it to the gateway.

The Gateway receives *M2* and decrypts it by executing $SDec(GrpKey_{GrpId}, M2)$ to obtain $GwEUI_{Gwi}||DevEUI_{EN_i}||GwSKey_{EN_i}$.Then, it calculates $h(GwSKey_{ENi}||DevEUI_{ENi})$ to generate a unique anonymous identifier for the end-node. Gw_i, also stores a maximum idle time (defined by the network administrator) *Max Idle Time* (MIT_{ENi}) for such EN_i to prevent storing data of devices that are not using that gateway or that devices that have not transmitted data in a period of time greater than (MIT_{ENi}). Finally, the gateway stores $\{GwDevId_{ENi}, GwSKey_{ENi}, MIT_{ENi}\}$ in its database for decrypting further messages sent by a particular end-node.

According to the specification, once the Join-Accept message was received, the end-node must derive session keys. At this point the End-Node calculates:

- $\boldsymbol{NAKey_{EN_i}}$=$NwkKey_{ENi} \otimes AppKey_{ENi}$
- $\boldsymbol{GwSKey_{ENi}}$=$aes\_cmac(NAKey_{EN_i}, h(DevEUI_{ENi}||DevNonce_{ENi}||JoinNonce_{ENi}||JoinEUI_{ENi}))$

to obtain the session key used to send messages to a particular Gateway. The $GwSKey_{ENi}$ is a 128-bit key. This key will be renewed on every Re-Join procedure according to the protocol described before. The key is assumed to be stored in a secure place with tamper proof mechanisms.

2) ROAMING SCENARIO

In case the LoRaWAN infrastructure is working on Roaming Scenario, the following considerations are in place, and the protocol for such scenario is shown in Figure 9.

According to the LoRaWAN backend specification [14] when an End-Node EN_i works over roaming the following additional steps are required once a Join Request has been dispatched. First, the Join Request arrives to *NS2* and it has to determine if it is acting as the *(hNS)* for the *ENi*. It also has to determine if it has been identified to work with *JS* which is identified by *JoinEUI*, if such is not the case, the process must terminate at this point. Otherwise, it has to perform a DNS lookup to identify the IP address of *JS*. In case *NS2* is not able to identify the *(hNS)*, it has to send a request that contains *DevEUI* to *JS* to retrieve such information. *JS* has to respond to such request either with a succesfull response containing the *NetID* of *NS1* if *NS2* belongs to authorized networks or with a No Roaming Agreement Response. Then, *NS2* performs a DNS lookup to obtain the IP Address of *NS1 (hNS)* by using the previously obtained *NetID* and also it sends a request *(ProfileReq)* containing the DevEUI to retrieve profile information of the device. Later, if the device is allowed for roaming *NS1* should inform to *NS2* through

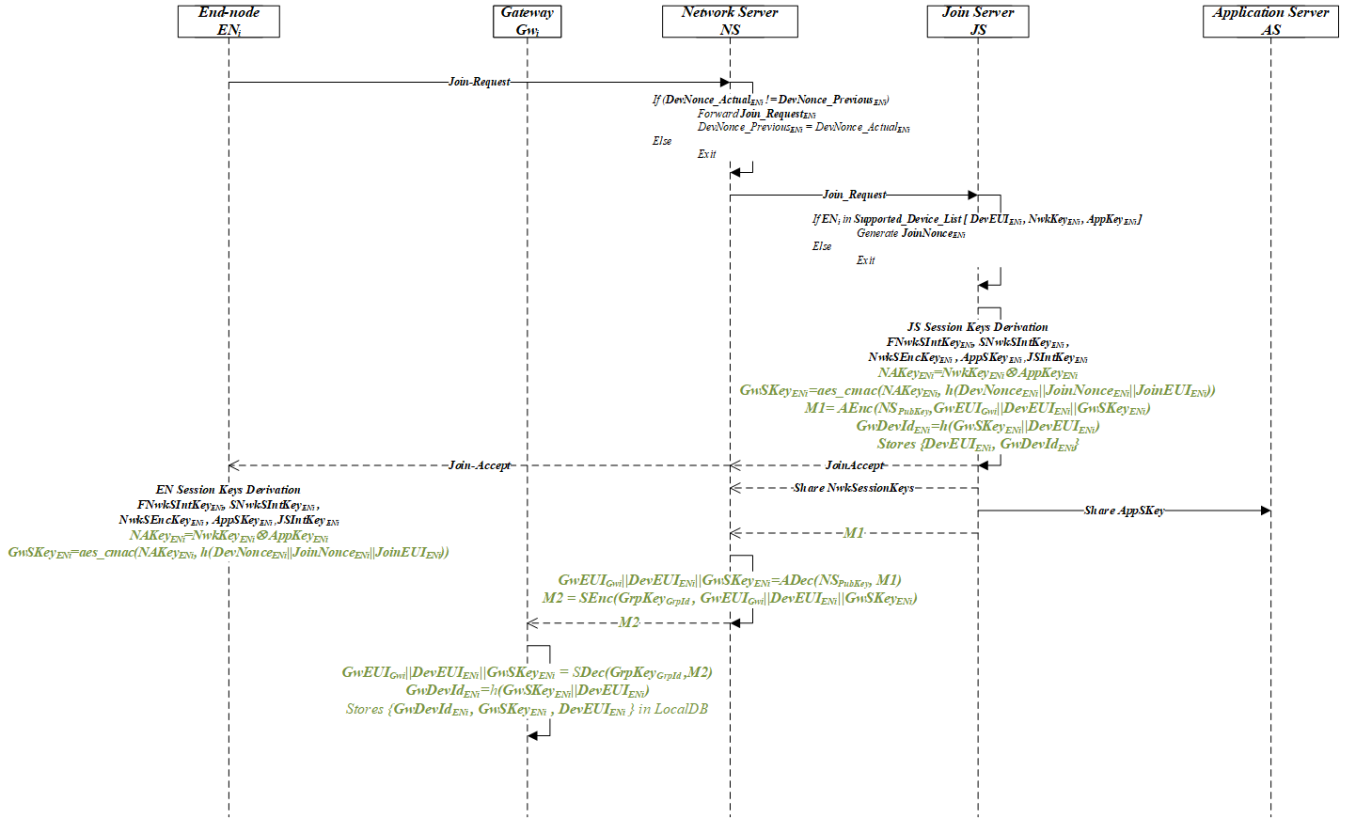

FIGURE 8. Gateway session key registration protocol home sceneraio.

a successfull notification *(ProfileAns)*. If the device has not been authorized a failure notification is forwarded. Once *NS2* received a successfull confirmation with handover roaming type, it has to start a new message request *(HRStartReq)* to *NS1* that contains the JoinRequest, MACVersion, ULMeta-Data, DevAddr, DLSettings, RxDelay, CFList and Device Profile Timestamp. *NS1* forwards the Join-Request to *JS* to start the *JS* Session Key Derivation process. During this process, the derivation of the new key $GwSKey_{EN_i}$ is the same as described in previous section. *JS* send an answer message *(HRStartAns)* to *NS2* containing the roaming activation status as well as Join-Accept response. The differences here compared to previos home scenario are that ***M1*** will be encrypted with the public key of ***M2*** and *M2* is message encrypted with the public key of *NS1*, as described below:

- $\boldsymbol{M1} = AEnc(NS1_{PubKey}, GwEUI_{Gw_i}||DevEUI_{EN_i}||GwSkey_{EN_i})$
- $\boldsymbol{M2} = AEnc(NS2_{PubKey}, ADec(NS1_{PubKey}, M1))$

Upon reception of *M2*, *NS2* calculates *M3* by executing $ADec(NS2_{PubKey}, M2)$ and then builds *M4* by executing $SEnc(GrpKey_{GrpId}), M3$, *M4* is then forwarded to the gateway.

Once Gw_i receives *M4*, it executes $SDec(GrpKey_{GrpId}), M4)$ to obtain $GwEUI_{Gw_i}||DevEUI_{EN_i}||GwSkey_{EN_i}$. Then, it calculates $GwDevId_{EN_i} = h(GwSkey_{EN_i}||DevEUI_{EN_i})$ and stores $\{GwDevId_{EN_i}, GwSkey_{EN_i}\}$ in tis LocalDB.

Finally, EN_i derives $GwSkey_{EN_i}$ in the same way as stated in the previous section (Home Scenario).

C. UPLINK MESSAGES THROUGH AUTHENTICATED GATEWAYS

The process for sending uplink messages through a registered gateway is described as follows. This procedure is executed after the OTAA Join-Procedure has been successfully acknowledge with a Join-Accept message. It applies for Unconfirmed Data Up Messages and is divided in two scenarios.

The first one applies when a end-node EN_i has joined (OTAA Activation) through a registered gateway Gw_i which has already been registered through the *fNS*. The second scenario applies when an end-node EN_i wants to send a message over a registered gateway but EN_i is not registered over that Gw_i. In our proposal a Gw_i must be registered over the LoRaWAN infraestructure before forwarding any message.

First of all, EN_i calculates all the following as part of the construction of the uplink message according to LoRaWAN 1.1 specification [13]:

- $\boldsymbol{MHDR}=Mtype||\boldsymbol{EJP}||Major$
- $\boldsymbol{FHDR}=DevAddr||FCtrl||FCnt||FOpts$
- $\boldsymbol{msg}=MHDR||FHDR||FPort||FRMPayload$
- $\boldsymbol{cmacS}=aes\_cmac(FNwkSIntKey_{ENi}, B1||msg)$
- $\boldsymbol{cmacF}=aes\_cmac(FNwkSIntKey_{ENi}, B0||msg)$
- $\boldsymbol{MIC}=cmacS[0..1]||cmacF[0..1]$
- $\boldsymbol{FRMPayload} = SEnc(AppSKey_{ENi}, Payload)$
- $\boldsymbol{PHYPayload}=MHDR||FHDR||FPort||FRMPayload$

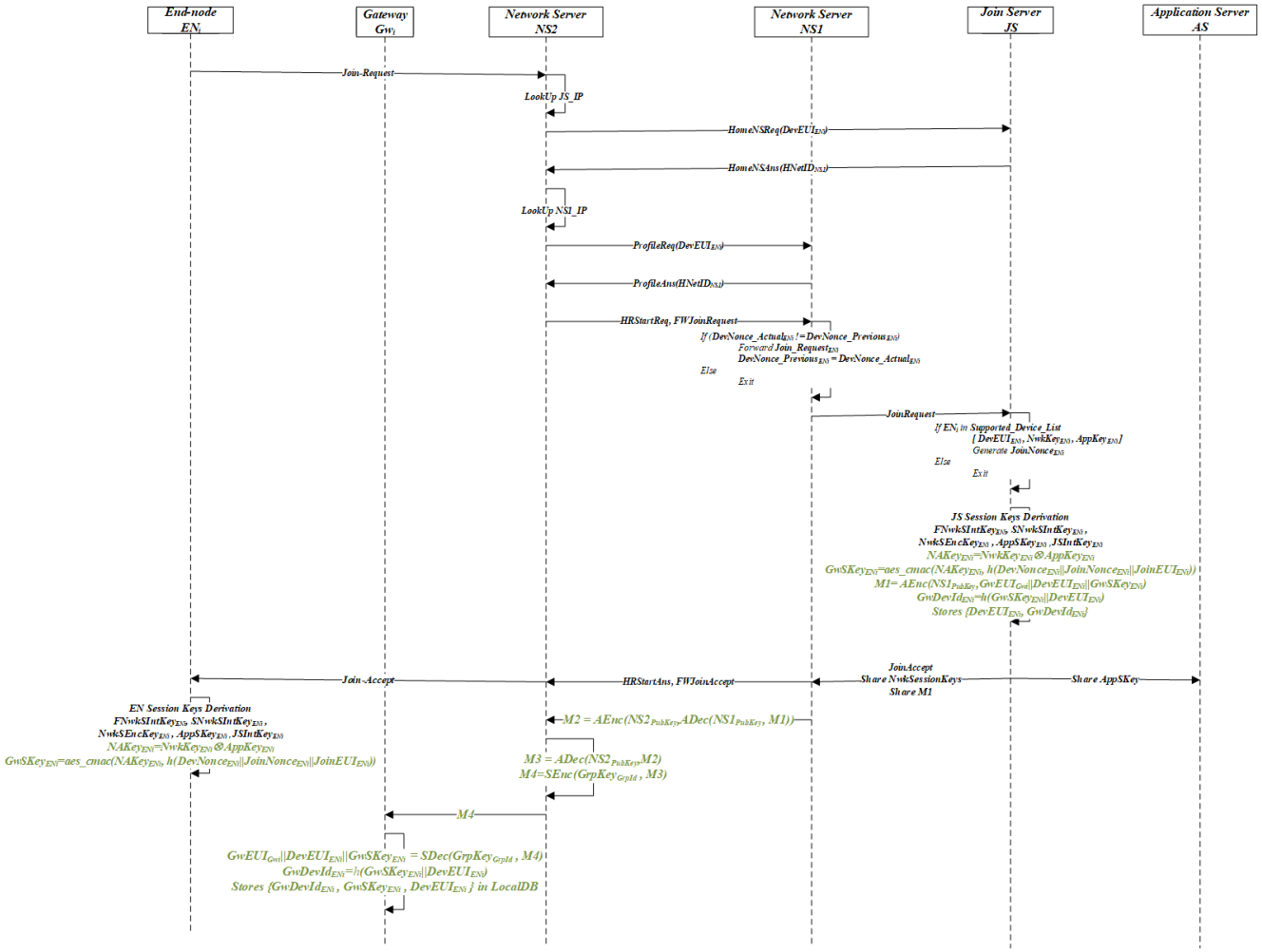

FIGURE 9. Gateway session key registration protocol roaming scenario.

Then, EN_i calculates $\boldsymbol{GwDevId_{ENi}}=h(GwSKey_{ENi}||DevEUI_{EN_i})$ which is a temporary anonymous identifier that depends on a session key previously established with a Join-Accept and changes with every message, it is 9 bytes long distributed as follows. The hashed parameter has been divided in 4 parts, the protocol will randomly take one of the parts (8 bytes) and will add a ninth byte to mark the corresponding portion sent.

For the **FHDR**, we propose to use dynamic Device Address to make sniffing harder. The new device Address (***DevAddr***) will be calculated as follow $\boldsymbol{DevAddr_{ENi}} = SEnc(\boldsymbol{GwDevId_{ENi}}, DevAddr)$. This parameter will use the full key $\text{GwDevId}_{\text{ENi}}$ generated and will be increased with every time an uplink message is sent.

Also, According to LoRaWAN 1.1 Specification [13] there are two unused (4..2) bits in MAC Header that are reserved for future use (RFU). This proposal uses thise bits so that EN_i defines the type of message to be built by creating ***EJP*** which will take the value $\boldsymbol{EJP = 0x01}$ to identify a secured type of message to be delivered through a registered gateway (Gw_i).

In addition, our proposal considers adding an integrity MIC for *FRMPayload*. EN_i calculates a new MIC after *FRMPayload* has been symmetrically ciphered. This new MIC is 4 bytes length as is calculated as follows $\boldsymbol{MIC\_Py}=aes\_cmac(AppSKey_{ENi},\ FRMPayload)[0..3]$ and will be used to validate, accept or decline a message if *FRMPayload* was tampered by malicious users.

Moreover, our proposal considers adding a 4-byte MIC for validating messages sent from EN_i to *Gwi*. This MIC is calculated as follows by EN_i $\boldsymbol{MIC\_P_{ENi}}=aes\_cmac(GwSKey_{ENi}, msg||GwDevId_{ENi}||FCntUp)[0..3]$.

Finally, once all previous components have been calculated, EN_i calculates $\boldsymbol{M1} = MHDR||FHDR||FPort||FRMPayload||MIC||\boldsymbol{MICPy}||\boldsymbol{MIC\_P_{EN_i}}$ and sends it to Gw_i.

1) PROTOCOL FOR SENDING UPLINK MESSAGES OVER AUTHENTICATED END-NODES AND GATEWAYS (UMOAEG)

The purpose of this protocol is to deliver messages over a Gw_i that has already been registered within LoRaWAN infrastructure by using an authenticated *Eni*.

Once Gw_i receives *M1*, it verifies if $GwDevId_{ENi}$ is in the *LocalDB* $GwDevId_{ENi}$, $GwSKey_{ENi}$. If such validation is true, it calculates $\boldsymbol{DevAddr} = SDec(\boldsymbol{GwDevId_{ENi}}, DevAddr_{ENi})$, extracts $GwSKey_{ENi}$ and calculates $\boldsymbol{MIC\_P_{ENi}}' = aes\_cmac(GwSKey_{ENi}, msg||GwDevId_{ENi}||FCntUp)[0..3]$ and compares against MIC_P_{ENi} from *M1*, if it matches, Gw_i builds $\boldsymbol{M2} = MHDR||FHDR||FPort||FRMPayload||MIC||\boldsymbol{MICPy}$ and calculates $\boldsymbol{M3}=SEnc(GrpKey_{GrpId}, M2)$ which is then forwarded to the Network Server *(NS)*.

Then, *NS* calculates $SDec(GrpKey_{GrpId}, \boldsymbol{M3})$ to obtain ***M2***. It then calculates ***cmacS*** and ***cmacF*** to validate ***MIC*** according to [13], if such validation is true ***M2*** is then forwarded to the Application Server *(AS)*.

Finally, after *AS* receives *M2*, it calculates $\boldsymbol{MIC\_Py}' = aes\_cmac(AppSKey_{ENi}, FRMPayload)[0..3]$ and validates against ***MIC_Py*** from *M2* to verify that FRMpayload has not been altered whilst in transit. If that validation was succesfull it then executes $SDec(AppSKey_{ENi}, FRMPayload)$ to obtain Payload in plain text and then decodes it; otherwise, *AS* aborts the process. The designed protocol is shown in Figure 10.

2) PROTOCOL FOR SENDING UPLINK MESSAGES OVER unAUTHENTICATED END-NODES AND GATEWAYS (UMOUEG)

The protocol designed can be seen in Figure 11 In this scenario, the purpose is to deliver an uplink message over an authenticated EN_i a registered Gw_i but the session key has not been delivered yet to Gw_i. First of all, once Gw_i receives *M1*, it verifies if $GwDevId_{ENi}$ is not in the *LocalDB* $GwDevId_{ENi}$, $GwSKey_{ENi}$. If so, it temporally stores M1,$GwDevId_{ENi}$,$\text{GwSKey}_{\text{ENi}}$ in a TempDB. Then, it extracts $DevEUI_{En_i}$ from LocalDB, calculates $\boldsymbol{M2}=SEnc(GrpKey_{GrpId}, GwDevId_{ENi}||GwEUI_{Gwi}||DevEUI_{ENi})$ and forwards it to *NS*.

Upon reception of*M2*, *NS* executes $SDec(GrpKey_{GrpId}, M2)$ to obtain $GwDevId_{ENi}||GwEUI_{Gwi}||DevEUI_{ENi}$ and then calculates $\boldsymbol{M3}=AEnc(Pubkey_{JS}, GwDevId_{ENi}||GwEUI_{Gwi}||DevEUI_{ENi})$ using asymmetric encryption with the public key of ***JS*** and forwards it.

Once ***JS*** receives *M3* it asymmetrically decrypts it by executing $ADec(Pubkey_{JS}, M3)$ to obtain $GwDevId_{ENi}||GwEUI_{Gwi}||DevEUI_{ENi}$ and then it validates if $GwDevId_{ENi}$ is in LocalDB and $DevEUI_{ENi}$ is in the Supported Device List of *JS*, if so it calculates the following:

- $\boldsymbol{NAKey_{ENi}}=NwkKey_{ENi} \otimes AppKey_{ENi}$
- $\boldsymbol{GwSKey_{ENi}}=aes\_cmac(NAKey_{ENi}, h(DevEUI_{ENi}||DevNonce_{ENi}||JoinNonce_{ENi}||JoinEUI_{ENi}))$
- $\boldsymbol{M4}=AEnc(Pubkey_{NS}, GwEUI_{Gwi}||DevEUI_{EN_i}||GwSKey_{EN_i}||GRANTED)$

On the other hand if there is no match *JS* calculates $\boldsymbol{M4}=AEnc(Pubkey_{NS}, GwEUI_{Gwi}||DevEUI_{EN_i}||UNAUTHORIZED)$. Then ***M4*** is sent back to *NS*.

NS receives *M4* and then asymmetrically decrypts it by executing $ADec(Pubkey_{NS}, M4)$ to obtain *M5* to calculate $\boldsymbol{M6}=SEnc(GrpKey_{GrpId}, M5)$ and then sents it back to Gw_i.

TABLE 6. Notations used in BAN logic.

Notation	Description
$X \mid\equiv Y$	X believes a statement Y
#(Y)	X is updated and fresh
$X \triangleleft Y$	X sees that Y
$X \mid\sim Y$	X once said the statement Y
$X \Rightarrow Y$	X controls that Y
$X \overset{K}{\longleftrightarrow} Y$	K is a secret shared key between X and Y
$\overset{K}{\longmapsto} X$	X has K as a public key
$\{Y\}_K$	Y is encrypted with K
$\langle Y\rangle_K$	Y is combined with K

Gw_i receives *M6* and symetrically dercrypts by executing $SDec(GrpKey_{GrpId}, M6)$ to obtain $MR = GwEUI_{Gw_i}'||DevEUI_{EN_i}'||GwSkey_{EN_i}'||STATUS$. Then, Gw_i validates if *MR* contains GRANTED response, if so, it then retrieves M1,$GwDevId_{EN_i}$,$GwSkey_{EN_i}$ from TempDB by using $GwSkey_{EN_i}$and calculates $\boldsymbol{MIC\_P_{ENi}}' = aes\_cmac(GwSKey_{ENi}, msg||GwDevId_{ENi}||FCntUp)[0..3]$. It compares if $\boldsymbol{MIC\_P_{ENi}}'$ is equal to $\boldsymbol{MIC\_P_{ENi}}$ obtained from M1, stores ($GwDevId_{EN_i}$, $GwSkey_{EN_i}$, $DevEUI_{EN_i}$), calculates $\boldsymbol{DevAddr} = SDec(\boldsymbol{GwDevId_{ENi}}, DevAddr_{ENi})$, builds $\boldsymbol{MP1}=MHDR||FHDR||FPort||FRMPayload||MIC||\boldsymbol{MICPy}$, calculates $\boldsymbol{MP2}=SEnc(GrpKey_{GrpId}, MP1)$ and forwards it to *NS* .

Then, *NS* calculates $SDec(GrpKey_{GrpId}, \boldsymbol{MP2})$ to obtain ***M2***. It then calculates ***cmacS*** and ***cmacF*** to validate ***MIC*** according to [13], if such validation is true ***M2*** is then forwarded to the Application Server *(AS)*.

Finally, after *AS* receives *M2*, it calculates $\boldsymbol{MIC\_Py}' = aes\_cmac(AppSKey_{ENi}, FRMPayload)[0..3]$ and validates against ***MIC_Py*** from *M2* to verify that FRMpayload has not been altered whilst in transit. If that validation was succesfull it then executes $SDec(AppSKey_{ENi}, FRMPayload)$ to obtain Payload in plain text and then decodes it; otherwise, *AS* aborts the process.

IV. SECURITY ANALYSIS

A. FORMAL ANALYSIS

In this section we demonstrate the security of the Gateway Registration Protocol by using BAN logic.

1) BAN LOGIC NOTATIONS

The following Table 6 presents the notations used for BAN logic.

2) BAN LOGIC RULES

The following are the rules of BAN logic:

1) Message meaning rule: $\frac{X \mid\equiv X \overset{K}{\longleftrightarrow} Z, X \triangleleft \{Y\}_K}{X \mid\equiv Z \mid\sim Y}$
2) Nonce verification rule: $\frac{X \mid\equiv \#(Y), X \mid\equiv Z \mid\sim Y}{X \mid\equiv Z \mid\equiv Y}$
3) Jurisdiction rule: $\frac{X \mid\equiv Z \Rightarrow Y, X \mid\equiv Z \mid\equiv Y}{X \mid\equiv Y}$
4) Freshness rule: $\frac{X \mid\equiv \#(Y)}{X \mid\equiv \#(Y, W)}$
5) Belief rule: $\frac{X \mid\equiv (Y, W)}{X \mid\equiv Y}$

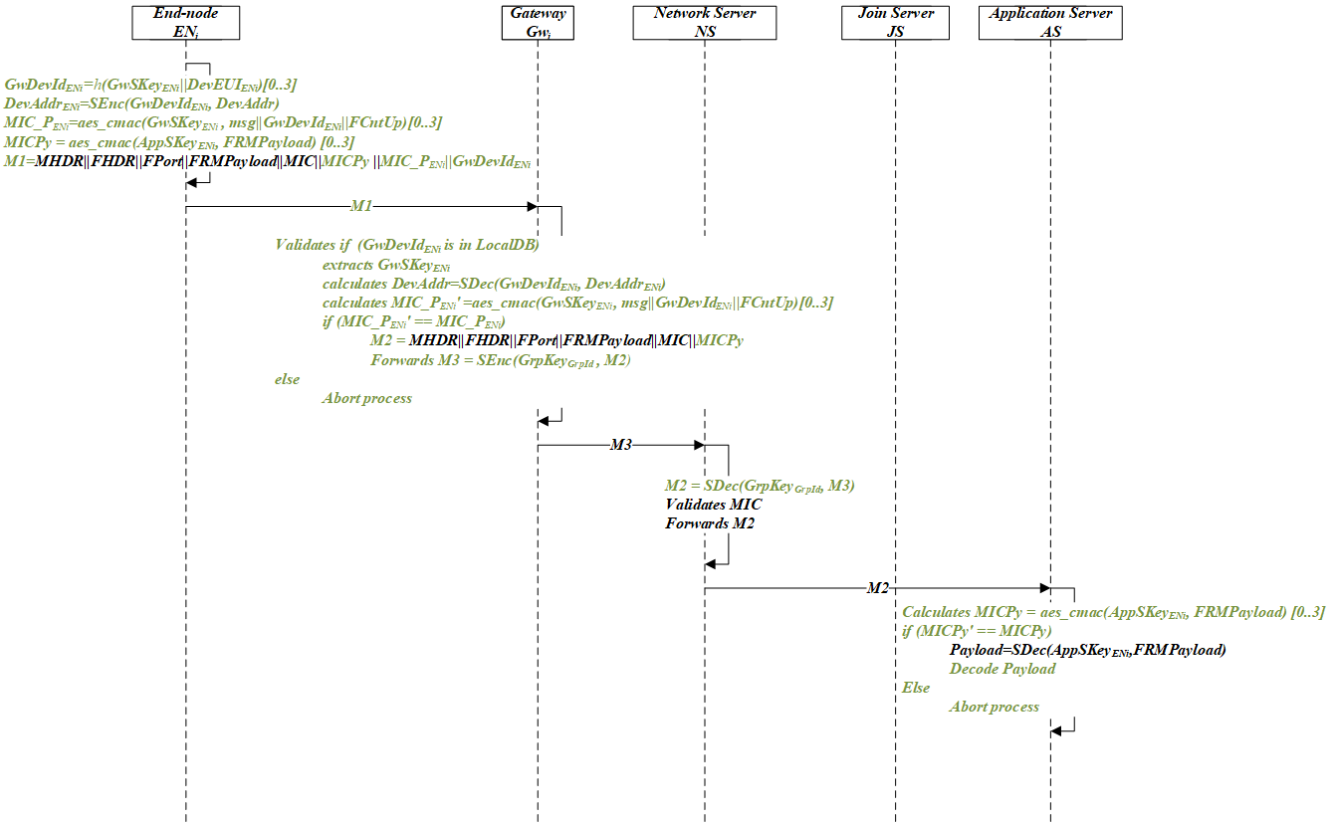

FIGURE 10. Uplink messages over authenticated end-node and gateway.

3) SECURITY GOALS

The following are the goals defined for the Gateway registration protocol:

Goal 1: $Gw_{\mathrm{i}} \mid\equiv (Gw_{\mathrm{i}} \xleftrightarrow{GrpKey_{\mathrm{GrpId}}} fNS)$

Goal 2: $fNS \mid\equiv (Gw_{\mathrm{i}} \xleftrightarrow{GrpKey_{\mathrm{GrpId}}} fNS)$

Goal 3: $Gw_{\mathrm{i}} \mid\equiv fNS \mid\equiv (Gw_{\mathrm{i}} \xleftrightarrow{GrpKey_{\mathrm{GrpId}}} fNS)$

Goal 4: $fNS \mid\equiv Gw_{\mathrm{i}} \mid\equiv (Gw_{\mathrm{i}} \xleftrightarrow{GrpKey_{\mathrm{GrpId}}} fNS)$

4) IDEALIZED FORMS OF MESSAGES

The idealized form of the messages of our protocol are shown below:

Msg1 :

$$Gw_{\mathrm{i}} \rightarrow fNS : \{GwSKa\}_{h(ID_{\mathrm{Ui}}||h(PW_{\mathrm{Ui}}))}, ID_{\mathrm{Ui}}, (RN1, GwEUI_{\mathrm{Gwi}})_{\mathrm{GwSKa}}$$

Msg2 :

$$fNS \rightarrow Gw_{\mathrm{i}} : \{RN1, RN2, Gw_{\mathrm{i}} \xleftrightarrow{GrpKey_{\mathrm{GrpId}}} fNS\}_{GwSKa}$$

Msg3 :$Gw_{\mathrm{i}} \rightarrow fNS$:

$$\{RN2, MICGw, h(GwEUI_{\mathrm{Gw_i}})\}_{Gw_{\mathrm{i}} \xleftrightarrow{GrpKey_{\mathrm{GrpId}}} fNS}$$

5) ASSUMPTIONS

The assumptions are listed below:

A1: $fNS \mid\equiv (Gw_{\mathrm{i}} \xleftrightarrow{h(ID_{\mathrm{Ui}})||h(PW_{\mathrm{Ui}})} fNS)$

A2: $fNS \mid\equiv \#(RN1)$

A3: $Gw_{\mathrm{i}} \mid\equiv (Gw_{\mathrm{i}} \xleftrightarrow{h(ID_{\mathrm{Ui}})||h(PW_{\mathrm{Ui}})} fNS)$

A4: $Gw_{\mathrm{i}} \mid\equiv \#(RN2)$

6) PROOF USING BAN LOGIC

1) According to **Msg1**, the following is obtained:

$$(S_1) : fNS \triangleleft GwSKa_{\mathrm{h(ID_{Ui} \,||\, h(PW_{Ui}))}}, ID_{\mathrm{Ui}}, (RN1, GwEUI_{\mathrm{Gw_i}})_{\mathrm{GwSKa}}$$

2) By using S_1 and A_1 with the message meaning rule, we obtain:

$$(S_2) : fNS \mid\equiv GWi \mid\sim \{(h(ID_{\mathrm{Ui}}||h(PW_{\mathrm{Ui}})), RN1, GwEUI_{\mathrm{Gw_i}}\}_{GwSKa}$$

3) Using S_2 and A_2, with the freshness rule, the following is obtained:

$$(S_3) : fNS \mid\equiv \#(GWSKa_{\mathrm{(h(ID_{Ui} \,||\, h(PW_{Ui}))}}), ID_{\mathrm{Ui}}, \{RN1, GwEUI_{\mathrm{Gw_i}}\}_{GwSKa}$$

4) By using S_1 and A_1 with the message meaning rule, we obtain:

$$(S_4) : fNS \mid\equiv GW_{\mathrm{i}} \mid\sim GWSKa$$

5) By using Nonce Verification Rules, S_3 and S_4, we obtain:

$$(S_5) : fNS \mid\equiv GW_{\mathrm{i}} \mid\equiv GWSKa$$

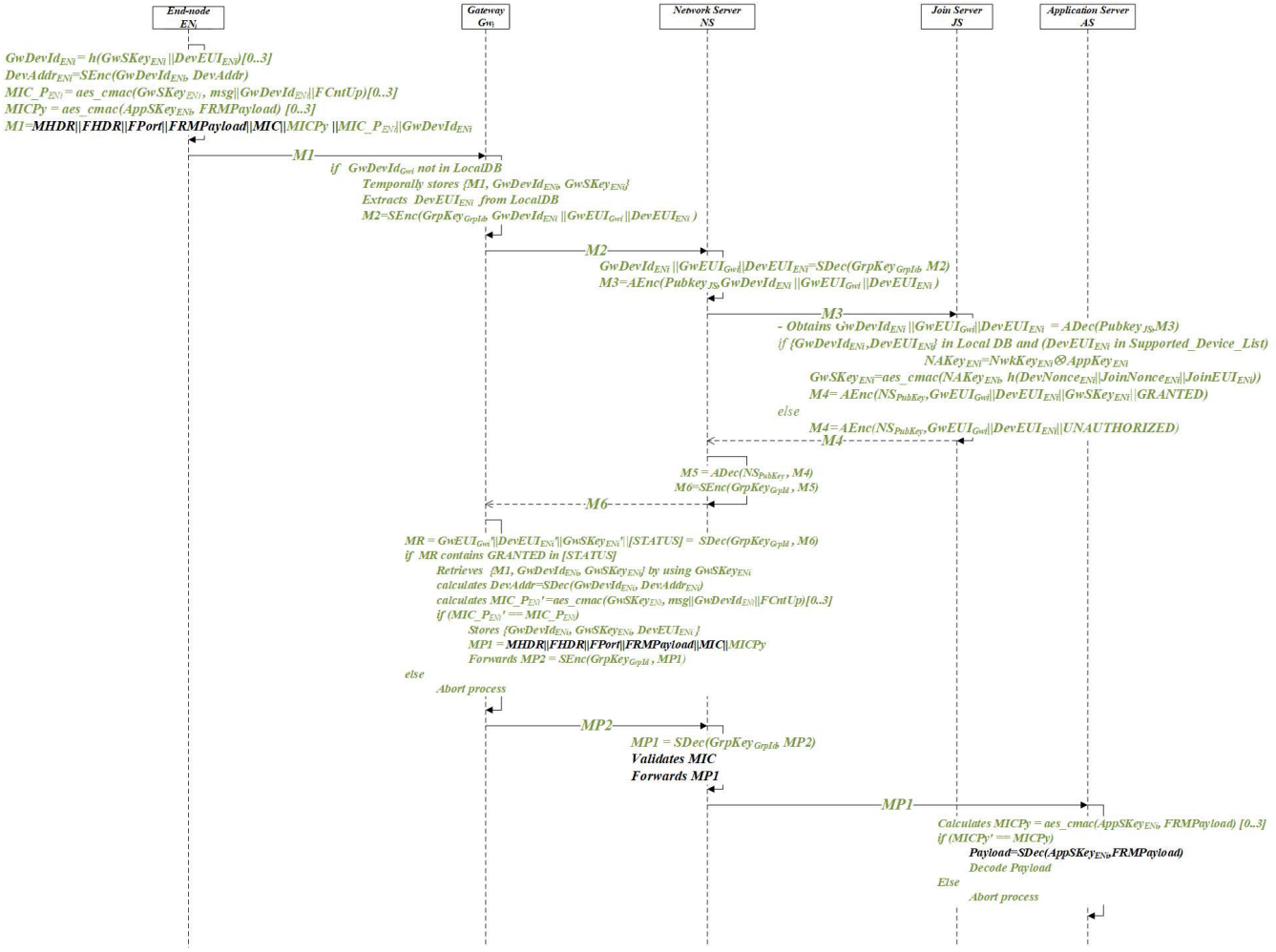

FIGURE 11. Uplink messages over unauthenticated end-node and gateway.

6) According to S_5 and Jurisdiction rule, we obtain:

$$(S_6) : fNS \mid\equiv GWSKa$$

7) According to **Msg2**, we obtain:

$$(S_7) : Gw_{\mathrm{i}} \lhd \{RN1, RN2, Gw_{\mathrm{i}} \overset{GrpKey_{\mathrm{GprId}}}{\longleftrightarrow} fNS\}_{GwSKa}$$

8) Using S_6 and S_7 with the message meaning rule, we obtained:

$$(S_8) : Gw_{\mathrm{i}} \mid\equiv fNS \mid\sim Gw_{\mathrm{i}} \overset{GrpKey_{\mathrm{GprId}}}{\longleftrightarrow} fNS$$

9) Using S_8, A_2 and A_4 with the nonce verification rule, we obtained:

$$(S_9) : Gw_{\mathrm{i}} \mid\equiv fNS \mid\equiv Gw_{\mathrm{i}} \overset{GrpKey_{\mathrm{GprId}}}{\longleftrightarrow} fNS \quad \textbf{(Goal3)}$$

10) Using S_9 and jurisdiction rule, the next is obtained:

$$(S_{10}) : Gw_{\mathrm{i}} \mid\equiv Gw_{\mathrm{i}} \overset{GrpKey_{\mathrm{GprId}}}{\longleftrightarrow} fNS \quad \textbf{(Goal1)}$$

11) By using the Key Generation Algorithm of the Protocol (Since $Gw_{\mathrm{i}} \overset{GrpKey_{\mathrm{GprId}}}{\longleftrightarrow} fNS$ is generated by fNS)

$$(S_{11}) : fNS \mid\equiv Gw_{\mathrm{i}} \overset{GrpKey_{\mathrm{GprId}}}{\longleftrightarrow} fNS \quad \textbf{(Goal2)}$$

12) According to **Msg3**, the following is obtained:

$$(S_{12}) : fNS \lhd \{RN2, MICGw, h(GwEUI_{\mathrm{Gw_i}})\}_{Gw_{\mathrm{i}} \overset{GrpKey_{\mathrm{GprId}}}{\longleftrightarrow} fNS}$$

13) By using S_{11}, S_{12}, and message meaning rule, we obtain:

$$(S_{13}) : fNS \mid\equiv GWi \mid\sim \left(RN2, MICGw, h(GwEUI_{\mathrm{Gw_i}})\right)$$

14) Using S_{13}, A_2 and A_4 with the nonce verification rule

$$(S_{14}) : GWi \mid\equiv fNS \mid\equiv (RN2, MICGw, h(GwEUI_{\mathrm{Gw_i}}))$$

15) By using the validation of the returned RN2, MICGw

$$(S_{15}) : fNS \mid\equiv GWi \mid\equiv Gw_{\mathrm{i}} \overset{GrpKey_{\mathrm{GprId}}}{\longleftrightarrow} fNS \quad \textbf{(Goal4)}$$

7) SCYTHER TOOL

Scyther is a tool that performs formal security analysis of protocols considering the assumption of perfect cryptography. It means that the adversary cannot learn from an encrypted message unless he possesses the key for decryption [36]. According to the authors, this tool helps finding problems when building protocols. This tool uses Security Protocol Description Language (SPDL), which has a programming syntax similar to C or Java.

Scyther is able to evaluate security properties such as; i. *Aliveness* ensures that partners are live, ii. *Weakagree* assures that a partner is communicating with each other rather than an intruder, iii. *Niagree* which means that the parties shall agree on the value of variables after a protocol has been executed, iv. *Nisynch* validates that everything is executed by triggers, occurs in order and contents are preserved, v. *SKR* refers to secrecy of session keys, vi. *Secret* refers to the secrecy of a particular parameter as stated in [36].

Scyther is developed over python and has a Graphic User Interface (GUI) and CLI interface, both of them can be used to analyze protocols and show claims. The results shown in tables 7 and 8 are taken from the GUI and are able to show a ''Failed'' statement in the ''Status'' column when there is a security issue and will display all attacks found with the help of a button that will launch a new window containing a graphic that denotes the attack. On the other hand, if everything goes well the ''Status'' column will show an ''OK'' statement combined with the ''No attacks'' words meaning that there are no attacks that affect the analyzed claim.

In terms of data types, Scyther is flexible and any type could be defined in order to represent a variable. It is important to clarify that Scyther does not analyze data types, it views the state of security of the whole protocol rather than checking for robustness of keys or algorithms used.

From the literature review, Scyther has been used to perform a formal analysis of the LoRaWAN protocol as described in [30]. In that work, authors prove the security of the OTAA Join-Procedure process by designing the protocol from scratch according to the specification. The results obtained showed that V1.0 is susceptible to attacks as there is a weak relation between the End-Node and the NS/AS particular during the join process. The tests performed were focused on Non-injective agreement and Non-injective synchronization.

To perform the analysis of the proposed protocols, we took the designs established in the previous section and translated them to the SPDL language following all the steps designed.

8) ANALYSIS OF GATEWAY REGISTRATION PROTOCOL

First of all, this protocol was coded including all variables as described in Figure 7. For this scenario, the Gw_i is authenticated against *NS* . Then *NS* , calculates a group key that includes all previous symmetric keys received from other gateways that have been registered already. Every time a gateway arrives or leaves this $GrpKey_{GrpId}$is recalculated. The Scyther analysis of this protocol is shown in Table 7. The secrecy of $GrpKey_{GrpId}$remains intact by showing no attacks, likewise all claims (Alive, Weakagree, Niagree, Nisynch) are marked with status *OK* showing that no attacks are possible. This validates that the proposed protocol is secure.

9) ANALYSIS OF GATEWAY SESSION KEY DERIVATION PROTOCOL

The Gateway Session Key Derivation Protocol was coded by including all variables involved during the OTAA procedure as described in the specification of LoRaWAN 1.1. This protocol is composed of three main roles: End-Node (Dev), Gateway, Network Server and Join Server. The results shown that Alive, Weakagree, Niagree and Nisynch are OK and are not susceptible to attacks. In addition, the secrecy of all the sessions keys generated is preserved among all the roles. The new introduced key $\boldsymbol{GwSKey_{ENi}}$ represented by $aes\_cmac(NAKey_{EN_i}, h(DevNonce_{ENi}||JoinNonce_{ENi}||JoinEUI_{ENi}))$ shows no attacks, meaning that the OTAA Join Procedure is not feasible to other attacks due to its inclusion. It is important to mention that this new session key preservers the same length (16 bytes) as other derived keys during Join Procedure.

This new key is shared to the Gateway through the Network Server, which uses a pre calculated group key ($GrpKey_{GrpId}$) to multicast this session key ($GwSkey_{EN_i}$) to other gateways that are connected to the same Network Server. Therefore, only the gateway or its group of gateways that belong to the same Network Server can decrypt messages sent by EN_i and then forward payloads to the backend infrastructure. The results of Scyther execution are displayed in Table 7.

10) ANALYSIS OF PROTOCOL UMOAEG

According to the LoRaWAN 1.1 specification once the Join Procedure has performed and End-Node device would be able to send data to the Application Server. As stated before, the design was translated to a SPDL file to reflect all interactions between the involved roles. In this case the participants were: End-Node (Dev), Gateway, Network Server (NS) and Application Server (AS).

All the variables used in the protocol where declared as String for testing purposes. String was defined as a userType variable as it is not a common data type of Scyther.

As shown in, Table 7 there are no potential vulnerabilities in the proposed protocol. It means that as long as an End-Node uses a valid *GwSkey*, a message will be delivered to the Application Server, otherwise, it will be discarded by the Gateway before sending it to the backend infrastructure.

The secrecy of session keys is preserved according to the results shown by Scyther as well as MIC and $MIC\_P_{EN_i}$ validation fields to protect the FRMPayload from bit-flipping attacks.

11) ANALYSIS OF PROTOCOL UMOUEG

In our proposed scenario, we have identified that if an EN_i has gone through a Join procedure using a different *Gateway.*

TABLE 7. Scyther results for proposed protocols I.

Protocol	Role	Claim	Status	Attack patterns
Gateway Registration Protocol	Gateway	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		Secret $GrpKey_{GrpId}$	*OK*	No attacks
	Network Server	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		Secret $GrpKey_{GrpId}$	*OK*	No attacks
Gateway Session Key Derivation Protocol	End-Node	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		SKR $GwSkey_{EN_i}$	*OK*	No attacks
	Gateway	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		Secret $GrpKey_{GrpId}$	*OK*	No attacks
		Secret $GwSkey_{EN_i}$	*OK*	No attacks
	Network Server	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		Secret $GrpKey_{GrpId}$	*OK*	No attacks
		Secret $GwSkey_{EN_i}$	*OK*	No attacks
	Join Server	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		Secret $GwSkey_{EN_i}$	*OK*	No attacks
UMOAEG Protocol	End-Node	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		Secret $GwSkey_{EN_i}$	*OK*	No attacks
		Secret *AppSKey*	*OK*	No attacks
	Gateway	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		Secret $GwSkey_{EN_i}$	*OK*	No attacks
		Secret *AppSKey*	*OK*	No attacks
	NS	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		Secret $GwSkey_{EN_i}$	*OK*	No attacks
		Secret *AppSKey*	*OK*	No attacks
	AS	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		Secret $GwSkey_{EN_i}$	*OK*	No attacks
		Secret *AppSKey*	*OK*	No attacks

It is possible to re-generate the $GwSkey_{EN_i}$and pass it to the Gateway so that it could deliver messages to the back-end infrastructure no matter if this is a newly authenticated Gateway over the platform. In case a rogue gateway aims to forward a message to the AS, it will not be able to determine the $GrpKey_{GrpId}$required to forward the payload to the *NS /fNS* .

For this scenario, there are five roles participating in the communication Dev, Gateway, NS, Join Server(JS) and Application Server (AS). Each of them is in charge of encrypting/decrypting particular parts of the message.

The results displayed by Scyther (see Table 8) showed that the implementation does not have potential attacks and it could be considered as a secure protocol. All claims are marked with the *OK* word and the *Verified Niagree, Nisynch*, *Alive, Weekagree* and session keys.

B. SECURITY ANALYSIS

This part examines the security of the proposed set of protocols by reviewing possible attacks [37].

TABLE 8. Scyther results for proposed protocols II.

Protocol	Role	Claim	Status	Attack patterns
UMOUAEG Protocol	End-Node	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		SKR *AppSKey*	*OK*	No attacks
		SKR $GwSkey_{EN_i}$	*OK*	No attacks
	Gateway	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		Secret $GrpKey_{GrpId}$	*OK*	No attacks
		Secret $GwSkey_{EN_i}$	*OK*	No attacks
	Network Server	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		Secret $GrpKey_{GrpId}$	*OK*	No attacks
		Secret $GwSkey_{EN_i}$	*OK*	No attacks
	Join Server	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		Secret *AppSKey*	*OK*	No attacks
		Secret $GwSkey_{EN_i}$	*OK*	No attacks

1) MAN IN THE MIDDLE ATTACK

This attack is not possible as the uplink messages dispatched are using secure encryption functions. When EN_i sends a message to Gw_i, it uses the symmetric session key (*GwSKey*) derived during the Join-Procedure. And when the Gw_i wants to send a message to fNS_i, it uses the symmetric key $GrpKey_{GrpId}$. Likewise, when fNS_i wants to communicate with Gw_i, it uses the calculated symmetric key $GrpKey_{GrpId}$. Using secure encryption functions, let proposed protocols to maintain confidentiality and integrity of messages.

2) REPLAY ATTACK

During the gateway registration protocol phase, random nonces are used to avoid replay attacks. Even if the attacker grabs the random nonce, he needs to have gateway credentials to perform a full registration procedure. Also, the attacker will not be able to generate valid messages to the gateway as the session key used to cipher it is calculated during Join and Rejoin procedures respectively.

3) PASSWORD GUESSING ATTACK

PW_{Ui} is not stored and is only known by the user in charge of performing registration procedure. A variant of it this value $h(PW_{Ui})$ is used to validate a user. It is important to consider that $h(.)$ is a one way has function that cannot reversed to obtain original credentials.

4) PRIVILEGED-INSIDER ATTACK

In the proposed solution, the network administrator (U_i) only have credentials for registering gateways and could not be able to capture other credentials because they are transmitted with a one-way hash function $h(PW_{Ui})$.

TABLE 9. LoRaWAN cryptographic operations.

LoRaWAN Operation	Entity	Cryptographic Operations						
		XOR	Hash	CMAC	SEnc	SDec	AEnc	ADec
Session key derivation	EN	-	-	-	5	-	-	-
	JS	-	-	-	5	-	-	-
Uplink message	EN	-	-	3	2	-	-	-
	NS	-	-	2	-	-	-	-
	AS	-	-	-	-	1	-	-

5) BRUTE FORCE ATTACK

The attacker might try to decrypt the uplink message generated by an end-node. However, the message is protected by symmetric key of 128-bit length that could be changed on demand.

6) SEPARATION OF RESPONSIBILITIES

A gateway (*Gw*) will only handle a pre-calculated temporary root key (*GwSKa*) and every end-node session key ($GwSKey_{ENi}$). A gateway will not be able to derive $GwSKey_{ENi}$ as it does not store parameters for such purpose.

C. CRYPTOGRAPHIC OPERATIONS

In order to determine a potential performance affectation, it is important to analyze and identify the number of additional cryptographical operations that will take place with the current proposed solution. This cryptographical operations comprise hashing, simple XOR, symmetric encryption, symmetric decryption, asymmetric encryption, and asymmetric decryption.

First of all, the current operation of the protocol already includes some cryptographic operations according to the specification [13] that are listed in the table below. The considered operations were taken from the Join-Procedure activation and the Uplink message delivery. Table 9 contains the operation name, the number of cryptographic operations,

TABLE 10. Table cryptographic operations of the proposed solutions.

Proposed Protocols	Entity	Cryptographic Operations XOR	Hash	CMAC	SEnc	SDec	AEnc	ADec
Gateway Registration	*Gw*	1	4	1	2	1	-	-
	fNS	1	5	1	1	2	-	-
GwSKey Derivation	*JS*	1	2	1	-	-	1	-
	fNS	-	-	-	1	-	-	1
	Gw	-	1	1	-	-	-	-
	EN	1	1	1	-	-	-	-
UMOAEG	*EN*	-	1	2	-	-	-	-
	Gw	-	-	1	1	-	-	-
	fNS	-	-	-	-	1	-	-
	AS	-	-	1	-	-	-	-
UMOUEG	*EN*	-	1	2	-	-	-	-
	GW	-	-	1	2	1	-	-
	fNS	-	-	-	1	2	1	1
	JS	1	1	1	-	-	1	1
	AS	-	-	1	-	-	-	-

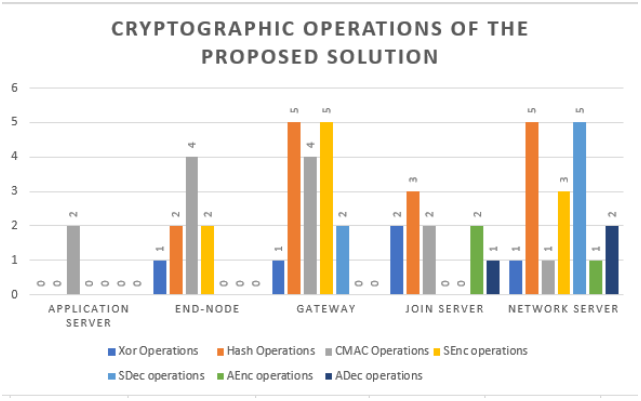

FIGURE 12. Cryptographic operations of the proposed solution per role.

the entity that performs such operation and the phase where that operation takes place (Join-Procedure or UplinkMSG delivery). For the analysis it is important not to overload the End-Node as it has limited computational and power resources.

The following table 10 shows the total number of additional cryptographic operations to be executed by every entity considering the new protocols proposed.

The following Figure 12 provides a summary of the additional effort to be made by all participant entities to implement the protocols of the proposed solution. According to the results shown in table 10 and figure 12, there are more encryption and decryption functions to be executed; however, none of them belong to the end-node. As mentioned before, the End-Node should not be overloaded as that is the entity with the lowest computational capacity, the other devices provide more computational resources so that the inclusion of new cryptographic functions would not affect its overall performance. Devices like the gateway are able to run over robust devices. The whole back-end infrastructure (Join-Server, Application Server and Newtork Server) are able to run over servers, virtual servers or containers in cloud infrastructures.

To have a better understanding on the impact over the End-Node the following Figures 13, 14 show a comparison

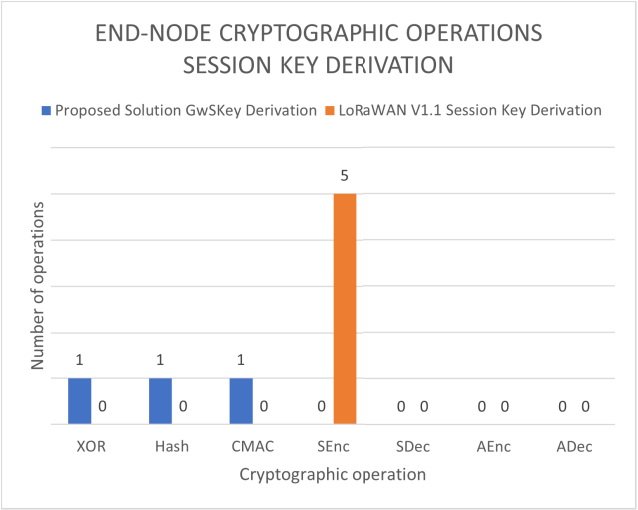

FIGURE 13. Cryptographic operations for end-node session key derivation.

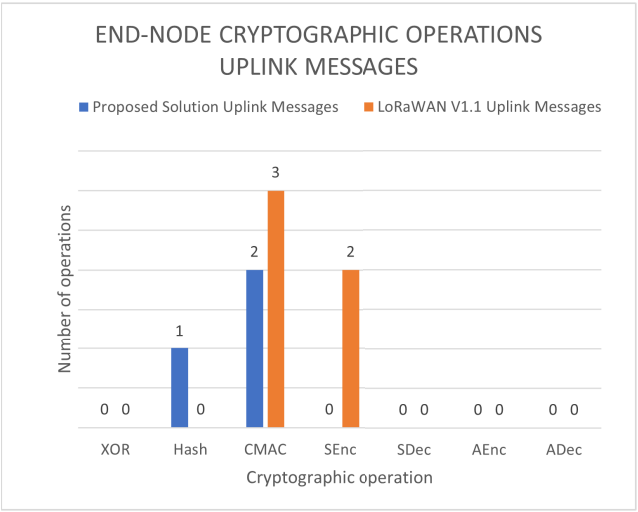

FIGURE 14. Cryptographic operations for end-node on uplink message delivery.

of the proposed solution with current LoRaWAN version in terms of cryptographic operations during the Session Key Derivation (Join-Procedure) and Uplink Message Delivery. In blue are all the new cryptographic functions added by the proposed solution whilst in orange are the current LoRaWAN cryptography operations. As showed one new type of operation is XOR. Also, another operation that comes from our proposal is hashing. CMAC operations refer to actions for calculating Message Integrity codes to guarantee message integrity. The proposed solution does not add new decryption functions or asymmetric operations.

The proposed protocols do not aim to implement new symmetric algorithms or to increase their encryption level (i.e. changing to AES-256). The solution is tied to the specification and although it will perform more cryptographic operations, their complexity will remain which means that the current computational resources would be enough to process the new operations.

TABLE 11. Parameters and its sizes.

Parameter	Description	Size (bytes)
RN	Random Nonces	8
RSK	Random Symmetric Key	32
$GwKey_{Gwi}$	Gateway Symmetric Key	32
GwSka	Gateway Symmetric Key Auth	32
GwInf	Gateway Auth Information	48
MReq	Message Registration Request	32
ID_{Ui}	User Id	16
PW_{Ui}	User password's	16
$GrpKey_{GrpId}$	Gateway Group Key	32
MICGw	AES CMAC	16
MA	MA	32
$GwEUI_{Gw_i}$	Gateway EUI	8
h(x)	One way Hash function SHA-256	32
MICPy	MIC Payload	4
$MICP_{EN_i}$	MIC Gateway Session Key	4
$GwDevId_{EN_i}$	Device Id with GwSkey	9
NS_{PubKey}	Network Server Public Key	256
JS_{PubKey}	Join Server Public Key	256
MHDR	MAC Header	1
DevAddr	Device Address	1
FCtrl	Frame Control	1
FCnt	Frame Counter	2
FPort	Frame Port	1
MIC	Message MIC	4

V. PERFORMANCE ANALYSIS OF THE PROPOSED PROTOCOLS

A. COMMUNICATION OVERHEAD

This section illustrates the communication overhead generated for Gateway Registration, GwSkey Derivation, UMOAEG and UMOUAEG protocols. This is an attempt to evaluate communication overhead over the aforementioned protocols. To calculate communication overhead, we assume that there are j number of EN_i and k number of Gw_i that might be connected to a LoRaWAN network infraestructure. The total number of additional bytes transmited for each protocol are the total number of bytes generated in each message of the previous listed protocols. The list of the parameters and its sizes is shown in table 11. The comparative analisys for communication overhead of the protocols is shown table 12. For this scenario, the communication overhead of the proposed protocols is computed as follows:

1) **Gateway Registration Protocol**
M1=MReq||ID_{Ui}||GwInf
M1=32*k + 16*k + 48*k
M1=96*k
M2=SEnc(GwSKa, $GrpKey_{GrpId}$||RN1'||RN2)
M2=SEnc(32,32+8+8)*k
M2=64*k
M3=MA||h($GwEUI_{Gw_i}$)
M3=32*k + 32*k
M3=64*k
Total_Overhead = M1+M2+M3 = 224*k

2) **Gw Session Key Derivation Protocol**
M1= AEnc(NS_{Pubkey},$GwEUI_{Gwi}$||$DevEUI_{EN_i}$ ||$GwSKey_{EN_i}$)
M1=AEnc(256, 8+8+16) * k
M1=256*k
M2=SEnc($GrpKey_{GrpId}$,$GwEUI_{Gwi}$||$DevEUI_{EN_i}$ ||$GwSKey_{EN_i}$)
M2=SEnc(32,8+8+16) * k
M2=32*k
Total_Overhead = M1+M2 = 288*k

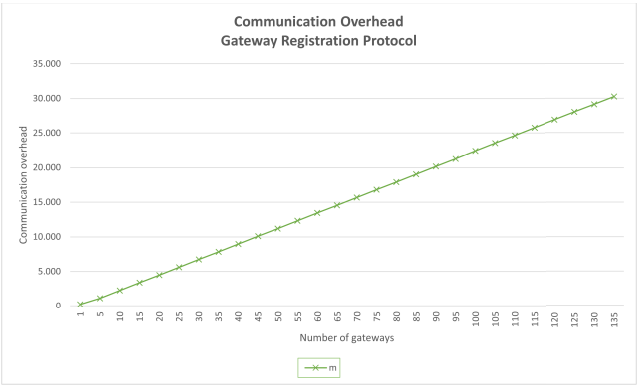

FIGURE 15. Gateway registration protocol communication overhead.

3) **UMOAEG Protocol**
M1 = MICPy||MIC_P_{EN_i}||$GwDevId_{EN_i}$
M1 = 4*j + 4*j + 9*j
M1 = 17*j
M2 = SEnc($GrpKey_{GrpId}$, $GwEUI_{Gwi}$||$DevEUI_{EN_i}$ ||$GwSKey_{EN_i}$)
M2 = SEnc(32,8+8+16)*k
M2 = 48*k
M3 = MICPY
M3 = 32*k
Total_Overhead = M1+M2+M3 = 17*j+80*k

4) **UMOUAEG Protocol**
M1=MICPy||MIC_P_{EN_i}||$GwDevId_{EN_i}$
M1 = 4*j + 4*j + 9*j
M1 = 17*j
M2 = 64*k
M3 = 256*k
M4 = 256*k
M6 = 64*k
MP1 = 17*k
MP2 = 32*k
Total_Overhead = M1+M2+M3+M4+M6+MP1+ MP2 = 17*j + 689*k

The total overhead for the Gateway Registration Protocol is 224*k this effect is shown in Figure 15. The total overhead for the Gateway Session Key Derivation Protocol is 288*k, this is shown in Figure 16. For UMOAEG protocol is 40*j+80*k (see Figure 17) and for UMOUAEF is 40*j + 689*k as described in Figure 18. Previous figures are estimates based on the effect of adding more nodes and gateways respectively for each of the designed protocols in this work. In all figures there is a directly proportional effect between the overhead generated and the number of devices added.

TABLE 12. Communications overhead per protocol.

Proposed Protocols	Communication Overhead Per Message							
GwSKey Derivation	**M1**	**M2**						**Total (Bytes)**
	256*j	32*k						288*k
Gw_i Registration	**M1**	**M2**	**M3**					**Total (Bytes)**
	96*k	64*k	64*k					224*k
UMOAEG	**M1**	**M2**	**M3**					**Total (Bytes)**
	17*j	48*k	32*K					17*j + 80*k
UMOUAEG	**M1**	**M2**	**M3**	**M4**	**M6**	**MP1**	**MP2**	**Total (Bytes)**
	17*j	64*k	256*k	256*k	64*k	17*k	32*k	17*j + 689*k

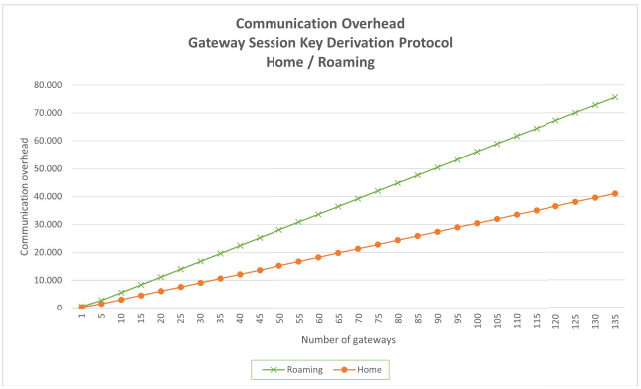

FIGURE 16. Gateway session key derivation protocol communication overhead.

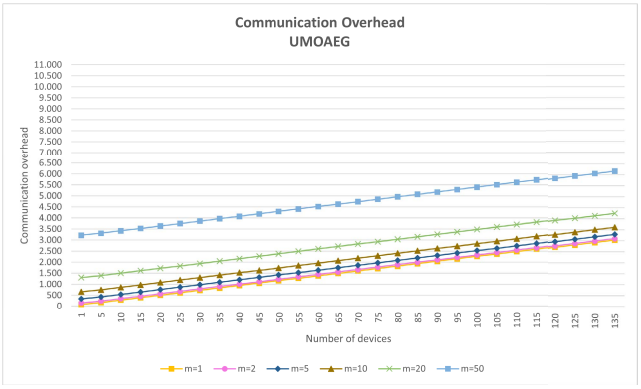

FIGURE 17. UMOAEG protocol communication overhead.

B. POWER CONSUMPTION ESTIMATION

In order to determine power consumption, it is important to clarify that it depends on the platform architecture where it will be deployed and the way the algorithm has been implemented as stated by [38].

To provide an estimate of the solution that we are proposing in this work, we will refer to one of the most common board for developing IoT prototypes and solutions which is Arduino. The authors in [39], perform a validation of the power used to cypher payloads using AES-128 in ECB,CBC and CTR modes, they have provided some statistics in terms of milliWatt (mW) consumed on the experiments performed. We are considering AES-128 in CTR mode as it is the algorithm used in LoRaWAN for encryption. For CMAC operations we will be considering the power used by an AES-128 encyrption operation.

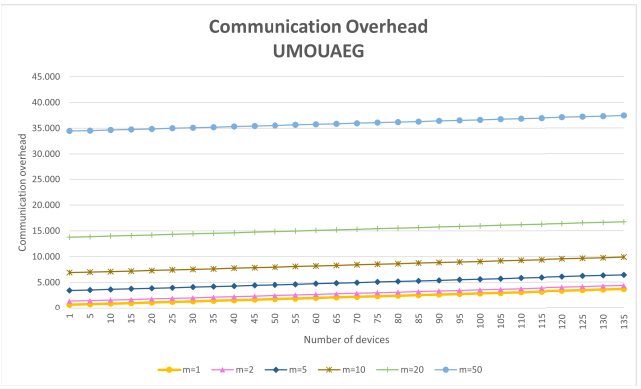

FIGURE 18. UMOUAEG protocol communication overhead.

TABLE 13. Values used to estimate power consumption.

Cryptographic operation	Power Consumption (mW)
AES-128 Encryption	41.7 [39]
AES-128 Decryption	42.6 [39]
XOR	1.1 [41]
HASH (SHA-1)	1.2 [40]

For estimating hashing power consumption, we will use the work described in [40]. The authors in their work use SHA-256 to validate crytographic hardware in IoT devices. The values obtained in their experiment will be used to establish an approximate value of consumption for hashing operations.

Based on the aforementioned approaches, we will consider the values obtained in their works and perform an estimation over the end-node because is the entity with the smallest amount of computational resources and power. The following Figure 19 shows the estimate power consumption expressed in milliWatts (mW) of the current LoRaWAN v1.1 Join procedure and uplink message procotol. The values taken to perform the estimation are shown in table 13.

According to the estimation the proposed solution would be adding an extra power consumption of 20% approximately on the end-node during the Join Procedure where the Gateway Session is generated. When sending uplink messages the proposed solution would add 40% extra power consumption. These values are estimates and intend to show the extra effort in terms of power needed by the end-node.

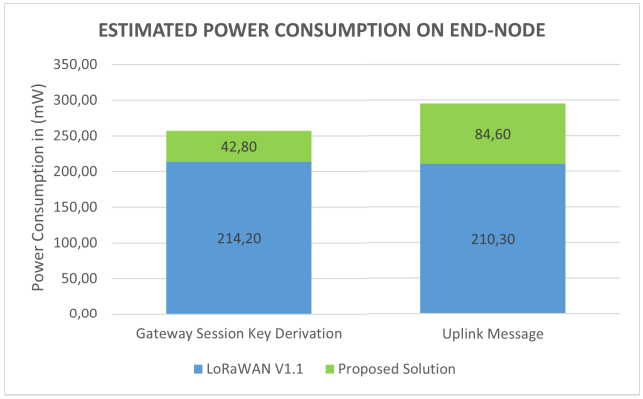

FIGURE 19. Power consumption in mW on the end-node for the proposed protocols.

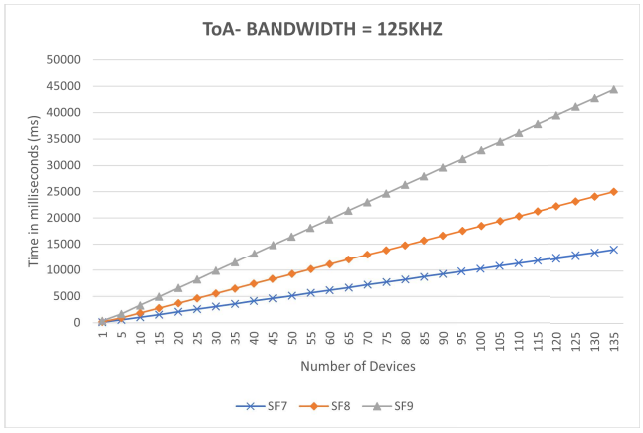

FIGURE 20. ToA for 125kHz considering the number of EN_i .

C. TIME OVER THE AIR (ToA) ESTIMATION

In order to determine the Time Over the Air for a particular payload of data being transmitted over a LoRaWAN network, we will use The Things Network LoRaWAN airtime calculator [42]. This calculator has four parameters to perform calculations, those are: Input Bytes, Spreading Factor (SF), Region, and Bandwidth. We have used this calculator to estimate ToA for the payload being generated by EN_i . The following table (see Table 14) shows the estimate ToA for a 19-byte payload (53-byte total packet size) generated with uplink messages over US915 region frequency. The Tail Size column includes *MIC* with 4 bytes, *MICPy* with 4 bytes, *MICPEni* with 4 bytes and $GwDevId_{EN_i}$ with 9 bytes. The chosen region does not support 250kHZ bandwidth [15]; therefore, that information is not shown in the following table.

The following Figures 20 and 21 shows the effect of ToA when more EN_i are presented within a LoRaWAN network. There is a directly proportional in time when more nodes send uplink messages through the network. It is important to remark that for US915 region there is a maximum dwell time (amount of time needed to transmit on a frequency) of 400ms per channels 0 to 63 but for channles from 64 to 71 there are no restrictions. Any payload above that time might not be delivered.

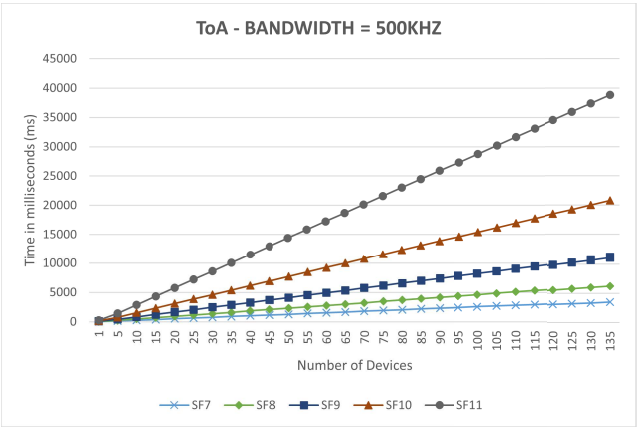

FIGURE 21. ToA for 500kHz considering the number of EN_i .

VI. DISCUSSION

The results of the formal security validation probe that the proposed protocol is secure in terms of security properties as validated by the Scyther tool. Although the results are favorable it is important to consider that when implementing these enhancements secure robust encryption algorithms must be in place (AES-128 at least) as well as a proper hashing algorithm (SHA-1 at least). Likewise, it is crucial to preserve the length of keys as the encryption algorithm in place suggested by the specification is AES-128; therefore, keys are restricted to have no more than 16 bytes.

Most of the reviewed works are focused on proposing new approaches for enhancing key management such as using blockchain which are the newest proposals. However, in terms of securing end-node to gateway communication there are no formal proposals although these issues are discussed in [10] and [19] in depth. The authors expose the need of mutual authentication protocols for securing communications between the aforementioned devices. To best of our knowledge those works are the only ones that point out the existence of this issue still in the new version of LoRaWAN.

Although other works like [20] or [16] exploit weaknesses in gateways, they do not show designs for possible implementations or new protocols that allow the identified gaps to be adequately mitigated.

Compared to other works our solution proposes a novel approach for authenticating a gateway within the back-end infrastructure as well as the interaction with the end-node by preserving the IoT premise of designing features that demand low energy consumption and low computational cost as well. The proposed approach does not include third-party services or infrastructure, on the contrary, it uses the elements defined in the specification and combines them to build protocols that allow ensuring the channel between the end-node and the gateway. Our proposed approach compared to the recent Basic Station [43] does not require the inclusion of a Certification Authority. In LoRa Basics Station, there are two scenarios that can be deployed one without authentication and the other which consists on using client and server authentication which involves configuring the service endpoint url,

TABLE 14. Time over the air for a single EN_i .

Header Size	Payload Size	Tail Size	Spread Factor (SF)	Bandwidth in kHz	ToA in ms
13	19	21	SF7	125	102.87
			SF8	125	184.8
			SF9	125	328.7
			SF10	125	Max Payload Size Exceeded
			SF11	125	Not supported by US915 region
			SF12	125	Not supported by US915 region
			SF7	500	25.7
			SF8	500	46.2
			SF9	500	82.2
			SF10	500	154.1
			SF11	500	287.7
			SF12	500	Max Payload Size Exceeded

a trusted certificate of the server's certification authority, a client token authentication or a client certificate with its private key. Our approach requires a pair of credentials (username and password) to be configured before automatically negotiating symmetric keys. Although our solution might be suceptible mistakes, it requires less steps than the LoRa Basics Station implementation based on TLS. We are looking forward to include these improvements by suggesting modifications to the current LoRaWAN specification, as there are fields that are not used like RFU which can be used to signal new cyphered packets created by the end-node according to the protocols proposed in this work. To the best of our knowledge, there are no other works that propose modifications to the current LoRaWAN specification.

VII. CONCLUSION, LIMITATIONS AND FUTURE WORK

The proposed solutions are secure according to the formal security verification analysis performed by Scyther. This tool allowed to identify potential security breaches that might affect the implementation of the protocol. Moreover, it allows to notice some concepts of freshness that are important when designing security protocols. During these formal verifications, we noticed that although the gateway acts as a ''simple packet forwarder'' it plays a crucial role in terms of security as it could affect the normal execution of the protocol.

This approach uses lightweight cryptographic functions that are not complex to be calculated and could be implemented over IoT devices without making further changes. It is expected that these new protocols demand more power consumption, but it will not be that significant to degrade the performance of the device. These functions are XOR, one-way hash which could be represented with SHA-1, symmetric encryption by preserving AES-128, CMAC calculation based on AES-128. As asymmetric encryption is not going to be implemented over the IoT, it will not degrade the performance of the overall solution as this will be executed by entities that have good computational resources.

One limitation of this protocols is that they have been tested by using Scyther, given that thare are many protocols in literature that have used this tool, some of them have been broken later. However, we have added BAN-Logic to prove that in terms of authentication between Gw_i and *NS /fNS* there are no potential issues.

For future work, we have considered to modify a LoRaWAN library for including these new changes over IoT devices based on Arduino hardware architecture to deploy the following proposed protocols. Likewise, perform a penetration testing over the Gateway Registration Protocol to prove its effectiveness. In addition, design a protocol to authenticate Class B beacon messages.

REFERENCES

[1] Y. Qian, Y. Jiang, J. Chen, Y. Zhang, J. Song, M. Zhou, and M. Pustišek, ''Towards decentralized IoT security enhancement: A blockchain approach,'' *Comput. Electr. Eng.*, vol. 72, pp. 266–273, Nov. 2018.

[2] C. Endress, G. Friedrich-Baasner, and D. Heim, ''From facets to a universal definition—An analysis of IoT usage in retail,'' in *Proc. Wirtschaftsinformatik*, 2019, pp. 1–12.

[3] R. Sharan Sinha, Y. Wei, and S.-H. Hwang, ''A survey on LPWA technology: LoRa and NB-IoT,'' *ICT Exp.*, vol. 3, no. 1, pp. 14–21, Mar. 2017.

[4] Statista Research Department. *IoT: Number of Connected Devices Worldwide 2012–2025 | Statista*. Accessed: Nov. 11, 2016. [Online]. Available: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

[5] M. Hung, ''Leading the IoT,'' Gartner, Stamford, CT, USA, Tech. Rep., 2017.

[6] J. J. Barriga, J. Sulca, J. L. León, A. Ulloa, D. Portero, R. Andrade, and S. G. Yoo, ''Smart parking: A literature review from the technological perspective,'' *Appl. Sci.*, vol. 9, no. 21, p. 4569, Oct. 2019.

[7] J. P. S. Sundaram, W. Du, and Z. Zhao, ''A survey on LoRa networking: Research problems, current solutions, and open issues,'' *IEEE Commun. Surveys Tuts.*, vol. 22, no. 1, pp. 371–388, 1st Quart., 2020.

[8] K. Mekkil, ''A comparative study of LPWAN technologies for large-scale IoT deployment,'' *ICT Exp.*, vol. 5, no. 1, pp. 1–7, 2019.

[9] M. A. Ertürk, M. A. Aydın, M. T. Büyükakkaşlar, and H. Evirgen, ''A survey on LoRaWAN architecture, protocol and technologies,'' *Future Internet*, vol. 11, no. 10, p. 216, Oct. 2019. [Online]. Available: https://www.mdpi.com/1999-5903/11/10/216

[10] J. J. Barriga A. and S. Guun Yoo, ''Internet of Things: A security survey review on long range wide area network (LoRaWAN),'' *J. Eng. Appl. Sci.*, vol. 14, no. 24, pp. 9774–9787, Sep. 2019.

[11] ''What makes Sigfox so secure for the Internet of Things,'' WND Group, Tech. Rep., Sep. 2017. [Online]. Available: https://www.wndgroup.io/2017/09/18/sigfox-security-internet-things/

[12] A. Lora, ''A technical overview of LoRa and LoRaWAN TM what is it?'' WND Group, Princes Risborough, U.K., Tech. Rep., 2015.

[13] ''LoRaWAN TM 1.1 specification,'' LoRa Alliance, Fremont, CA, USA, Tech. Rep., 2017, p. 97331. [Online]. Available: https://lora-alliance.org/wp-content/uploads/2020/11/lorawantm_specification_v1.1.pdf

[14] A. Lora, *Ts2-1.1.0 LoRaWAN Backend Interfaces Specification*, LoRa Alliance, Fremont, CA, USA, 2020, p. 85.

[15] L. A. T. C. R. P. Workgroup, *Rp002-1.0.3 LoRaWAN Regional Parameters*, LoRa Alliance, Fremont, CA, USA, May 2021.

[16] X. Yang, E. Karampatzakis, C. Doerr, and F. Kuipers, ''Security vulnerabilities in LoRaWAN,'' in *Proc. IEEE/ACM 3rd Int. Conf. Internet Things Design Implement. (IoTDI)*, Apr. 2018, pp. 129–140.

[17] I. Butun, N. Pereira, and M. Gidlund, "Analysis of LoRaWAN v1.1 security," in *Proc. 4th ACM MobiHoc Workshop Experiences Design Implement. Smart Objects (SMARTOBJECTS)*, 2018, pp. 1–6.
[18] D. Basu, T. Gu, and P. Mohapatra, "Security issues of low power wide area networks in the context of LoRa networks," 2020, *arXiv:2006.16554*.
[19] O. Magnusson, R. Teodorsson, J. Wennerberg, and S. A. Knoph, "A survey on attacks and defences on LoRaWAN gateways," in *Decision Support Systems and Industrial IoT in Smart Grid, Factories, and Cities*. Hershey, PA, USA: IGI Global, Jun. 2021, pp. 19–38.
[20] L. Simon Laufenberg, "Impersonating LoRaWAN gateways using semtech packet forwarder," 2019, *arXiv:1904.10728*.
[21] F. L. Coman, K. M. Malarski, M. N. Petersen, and S. Ruepp, "Security issues in Internet of Things: Vulnerability analysis of LoRaWAN, sigfox and NB-IoT," in *Proc. Global IoT Summit (GIoTS)*, Jun. 2019, pp. 1–6.
[22] E. Aras, G. S. Ramachandran, P. Lawrence, and D. Hughes, "Exploring the security vulnerabilities of LoRa," in *Proc. 3rd IEEE Int. Conf. Cybern. (CYBCONF)*, Jun. 2017, pp. 1–6.
[23] M. Ralambotiana, "Key management with a trusted third party using LoRaWAN protocol: A study case for E2E security," Skolan för Elektroteknik och Datavetenskap (EECS), Stockholm, Sweden, Tech. Rep., 2018, p. 78.
[24] S.-Y. Gao, X.-H. Li, and M.-D. Ma, "A malicious behavior awareness and defense countermeasure based on LoRaWAN protocol," *Sensors*, vol. 19, no. 23, p. 5122, Nov. 2019.
[25] M. Van Leent, "An improved key distribution and updating mechanism for low power wide area networks (LPWAN)," M.S. thesis, Fac. Governance Global Affairs, Univ. Leiden, Leiden, The Netherlands, 2017.
[26] J. Kim and J. Song, "A dual key-based activation scheme for secure LoRaWAN," *Wireless Commun. Mobile Comput.*, vol. 2017, Nov. 2017, Art. no. 6590713.
[27] R. Sanchez-Iborra, J. Sánchez-Gómez, S. Pérez, P. Fernández, J. Santa, J. L. Hernández-Ramos, and A. F. Skarmeta, "Enhancing LoRaWAN security through a lightweight and authenticated key management approach," *Sensors*, vol. 18, no. 6, p. 1833, Jun. 2018.
[28] J. Han and J. Wang, "An enhanced key management scheme for LoRaWAN," in *Lecture Notes Computer Science (Including Subseries Lecture Notes Artifical Intelligent Lecture Notes Bioinformatics)*. Cham, Switzerland: Springer, 2018, pp. 407–416.
[29] V. Ribeiro, R. Holanda, A. Ramos, and J. J. P. C. Rodrigues, "Enhancing key management in LoRaWAN with permissioned blockchain," *Sensors*, vol. 20, no. 11, p. 3068, May 2020.
[30] M. Eldefrawy, I. Butun, N. Pereira, and M. Gidlund, "Formal security analysis of LoRaWAN," *Comput. Netw.*, vol. 148, pp. 328–339, Jan. 2019.
[31] T. C. Dönmez and E. Nigussie, "Security of join procedure and its delegation in LoRaWAN v1.1," in *Proc. 15th Int. Conf. Mobile Syst. Pervasive Comput. (MobiSPC)/13th Int. Conf. Future Netw. Commun. (FNC)* (Procedia Computer Science), vol. 134, A. Yasar and E. M. Shakshuki, Eds. Gran Canaria, Spain: Elsevier, 2018, pp. 204–211, doi: 10.1016/j.procs.2018.07.202.
[32] J. Kim and J. Song, "A simple and efficient replay attack prevention scheme for LoRaWAN," in *Proc. the 7th Int. Conf. Commun. Netw. Secur. (ICCNS)*, 2017, pp. 32–36.
[33] S. Na, D. Hwang, W. Shin, and K.-H. Kim, "Scenario and countermeasure for replay attack using join request messages in LoRaWAN," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, 2017, pp. 718–720.
[34] S. M. Danish, M. Lestas, H. K. Qureshi, K. Zhang, W. Asif, and M. Rajarajan, "Securing the LoRaWAN join procedure using blockchains," *Cluster Comput.*, vol. 23, pp. 1–16, Feb. 2020.
[35] M. Ingham, J. Marchang, and D. Bhowmik, "IoT security vulnerabilities and predictive signal jamming attack analysis in LoRaWAN," *IET Inf. Secur.*, vol. 14, no. 4, pp. 368–379, 2020.
[36] C. J. F. Cremers, "Scyther: Semantics and verification of security protocols," Ph.D. dissertation, Dept. Math. Comput. Sci., Univ. Eindhoven, Eindhoven, The Netherlands, 2006.
[37] S. G. Yoo and J. J. Barriga, "Privacy-aware authentication for Wi-Fi based indoor positioning systems," in *Applications and Techniques in Information Security*, L. Batten, D. S. Kim, X. Zhang, and G. Li, Eds. Singapore: Springer, 2017, pp. 201–213.
[38] O. Khomlyak, "An investigation of lightweight cryptography and using the key derivation function for a hybrid scheme for security in IoT," M.S. thesis, Blekinge Inst. Technol., Stockholm, Sweden, 2017.
[39] L. E. Kane, J. J. Chen, R. Thomas, V. Liu, and M. Mckague, "Security and performance in IoT: A balancing act," *IEEE Access*, vol. 8, pp. 121969–121986, 2020.
[40] P. Kietzmann, L. Boeckmann, L. Lanzieri, T. C. Schmidt, and M. W. Ahlisch, "A performance study of crypto-hardware in the low-end IoT," in *Proc. Int. Conf. Embedded Wireless Syst. Netw.*, 2021, pp. 79–90. [Online]. Available: https://github.com/
[41] M. Kumar, "Single bit full adder design using 8 transistors with novel 3 transistors XNOR gate," *Int. J. VLSI Des. Commun. Syst.*, vol. 2, no. 4, pp. 47–59, Dec. 2011.
[42] T. T. Industries. (2016). *The Things Network—LoRaWAN Airtime Calculator*. [Online]. Available: https://www.thethingsnetwork.org/airtime-calculator
[43] L. Alliance. *LoRa Basics Station | Developer Portal*. Accessed: Feb. 2, 2022. [Online]. Available: https://lora-developers.semtech.com/build/software/lora-basics/lora-basics-for-gateways

JHONATTAN J. BARRIGA received the M.Sc. degree in computer forensics and systems security from the University of Greenwich, London, U.K. He is currently pursuing the Ph.D. degree in computer science working on IoT Security topics. He was worked for IBM as an IT Architect, from 2007 to 2010. From 2012 to 2012, he worked at TATA Consultancy Services as an Information Security Coordinator. He is currently an Assistant Professor with the Departamento de Informática y Ciencias de la Computación, Escuela Politécnica Nacional, Quito, Ecuador. He is also a Systems Engineer (ESPE). His research interests include malware, penetration testing, trojans, secure coding, security architecture, and forensic computing. He is a Certified Ethical Hacker from EC-Council.

SANG GUUN YOO (Senior Member, IEEE) received the Ph.D. degree from the Department of Computer Science and Engineering, Sogang University, Seoul, South Korea, in 2013. He is currently a Professor with the Departamento de Informática y Ciencias de la Computación, Escuela Politécnica Nacional. He is one of the Co-Founder of ExtremoSoftware (Microsoft Gold Certified Partner), where worked as the CTO, from 2001 to 2005. From 2005 to 2007, he collaborated as a Professor with the Department of Computer Science and Multimedia, International University of Ecuador. From 2006 to 2007, he also worked as an IT Consultant for the Army Intelligence Agency of Ecuador. He also had the opportunity to work as a Chief Research Engineer at LG Electronics, Republic of Korea.