Formal Security Analysis of LoRaWAN

Mohamed Eldefrawy[a], Ismail Butun[a], Nuno Pereira[b], Mikael Gidlund[a]

[a] *Information Systems and Technology, Mid Sweden University, Sundsvall, Sweden*
[b] *School of Engineering (DEI/ISEP), Polytechnic of Porto (IPP), Porto, Portugal*

Abstract

Recent Low Power Wide Area Networks (LPWAN) protocols are receiving increased attention from industry and academia to offer accessibility for Internet of Things (IoT) connected remote sensors and actuators. In this work, we present a formal study of LoRaWAN security, an increasingly popular technology, which defines the structure and operation of LPWAN networks based on the LoRa physical layer. There are previously known security vulnerabilities in LoRaWAN that lead to the proposal of several improvements, some already incorporated into the latest protocol specification. Our analysis of LoRaWAN security uses Scyther, a formal security analysis tool and focuses on the key exchange portion of versions 1.0 (released in 2015) and 1.1 (the latest, released in 2017). For version 1.0, which is still the most widely deployed version of LoRaWAN, we show that our formal model allowed to uncover weaknesses that can be related to previously reported vulnerabilities. Our model did not find weaknesses in the latest version of the protocol (v1.1), and we discuss what this means in practice for the security of LoRaWAN as well as important aspects of our model and tools employed that should be considered. The Scyther model developed provides realistic models for LoRaWAN v1.0 and v1.1 that can be used and extended to formally analyze, inspect, and explore the security features of the protocols. This, in turn, can clarify the methodology for achieving secrecy, integrity, and authentication for designers and developers interested in these LPWAN standards. We believe that our model and discussion of the protocols security properties are beneficial for both researchers and practitioners. To the best of our

---

Email address: `mohamed.eldefrawy@miun.se`, `ismail.butun@miun.se`, `nap@isep.ipp.pt`, `mikael.gidlund@miun.se` (Mikael Gidlund)

Preprint submitted to Elsevier Computer Network *November 15, 2018*

knowledge, this is the first work that presents a formal security analysis of LoRaWAN.

Keywords:
LoRaWAN, IoT, Scyther Verification.

---

1. Introduction

The proliferation of Internet-connected sensor and actuator devices embedded in everyday objects (called "things") are shaping an ever-growing Internet of Things (IoT). We can find IoT applications in many areas, from home automation systems, industrial processes control, pollutant detection or smart metering, just to name a few examples. To build these IoT applications, developers have many connectivity options, such as IEEE 802.15.4, Bluetooth or IEEE 802.11 for short and medium range or LTE for long range. Another increasingly attractive option for deploying IoT applications are Low Power Wide Area Networks (LPWAN) protocols, as they can cover distances of several kilometres with minimal infrastructure. In fact, the popularity of LPWAN lead to the emergence of a number of competing technologies, such as Sigfox, LoRa, and NB-IoT.

In this work, we focus on LoRa, which is a commonly accepted LPWAN protocol, deployed all around the world. LoRa is particularly interesting due to the openness of its higher layer specifications - LoRaWAN, and for the wide availability of low-cost devices. LoRa is also the only technology allowing to build private LPWAN networks, Vangelista et al. (2015).

LoRa is a proprietary physical layer protocol that facilitates low-power and long-distance communication up to 20 Km by using Chirp Spread Spectrum (CSS) modulation technique. LoRaWAN is the upper layer protocol based on LoRa in which the structure and operation of the entire system are defined. LoRaWAN went through several iterations and refinements, and the latest version of the specification (1.1) was recently released in October 2017 Alliance (2017). LoRaWAN v1.1 was a major step forward in the specification and introduced a number of security-related features and improvements. Due to its recent release, the security of this version of LoRaWAN still has very little scrutiny, while there are several known vulnerabilities in previous specifications of LoRaWAN (see Section II). These vulnerabilities were found by inspection of the protocol, based on the researcher's expertise. To our knowledge, no formal verification of the protocol was made previously, so we set out to formally analyze LoRaWAN v1.0, which is still the most widely deployed version of the protocol Delbruel et al. (2017) and also the latest version (v1.1). We have reported some of our early findings in Butun et al. (2018), however this paper presents all the detailed security analysis and results along with comprehensive discussions.

To perform the protocol verification presented in this paper, we developed a model of LoRaWAN for the Scyther automatic protocol verification tool. Our model allows Scyther to show that v1.0 is vulnerable due to a lack of synchronization between communicating parties. This vulnerability reported by Scyther is related to attacks previously reported by researchers Tomasin et al. (2017); Na et al. (2017), which is interesting as it illustrates that our Scyther model can find practical vulnerabilities in LoRaWAN. We then build a Scyther model for v1.1 and this model shows that the latest version of LoRaWAN no longer suffers from this vulnerability. Furthermore, the model shows that it can enforce several relevant security claims which we describe in detail in Section III. We believe that these tools, models and the discussion of the security properties of the several versions of LoRaWAN is interesting for practitioners using the protocol and researchers trying to develop extensions and improvements.

The remainder of this paper is organized follows. Section II overviews the related background on LoRaWAN and protocol verification. Section III presents our models for LoRaWAN and the security claims, as well discussing their implications. In Section-IV, we discuss several security still open. Finally, conclusions and future work are presented in Section V.

2. Background

In this paper, we are interested in studying LoRaWAN. In this section, we will start by providing some details of the protocol and later, we overview some background related to automated protocol verification. The used notations in this manuscript are summarized in Table 1

2.1. LoRaWAN

In LoRaWAN, the network is composed of end-devices (ED) that are connected with a single hop to one or more Gateways which, in turn, forward packets to the Network Server (NS) through a back-haul network using IP protocols.

Notations	Description
ABP	Activation By Personalization
AES	Advanced Encryption Standard
$Alive$	Aliveness claim (Scyther)
$AppEUI$	Application Unique Identifier
$AppKey$	Application Key
AS	Application Server
EAP	Extensible Authentication Protocol
ED	End-Devices
$FNwkSIntKey$	Forwarding Network Session Integrity Key
IoT	Internet of Things
$JoinEUI$	Join Server Unique Identifier
$JoinNonce$	Join Server Nonce (random)
JS	Join Server
$LoRaWAN$	Long RangeWide Area Networks
$LPWAN$	Low Power Wide Area Networks
MIC	Message Integrity Code
$MITM$	Man-In-The-Middle
$NBIoT$	Narrowband-IoT
$NETID$	Network Identifier
$NwkSEncKey$	Network Session Encryption Key
$Nisynch$	Non-injective Synchronization claim (Scyther)
$Niagree$	Non-injective Agreement claim (Scyther)
NS	Network Server
$OTAA$	Over The Air Activation
PKI	Public Kay Infrastructure
SKR	Key Security claim (Scyther)
$SNwkSIntKey$	Serving Network Session Integrity Key

Table 1: Notations used for this paper

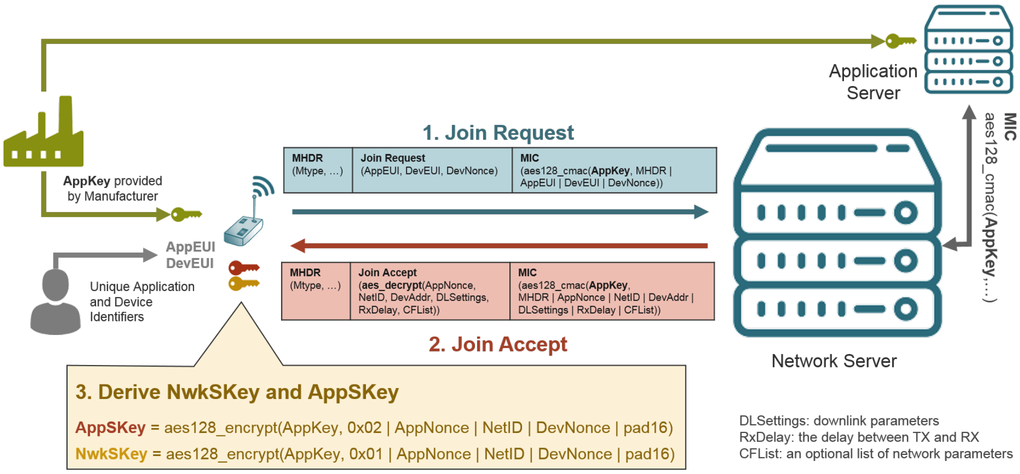


Figure 1: LoRaWAN v1.0 Over the Air Activation (OTAA) procedure

LoRaWAN defines that the secrecy and integrity of data payloads transmitted in the network are secured by employing well-known symmetric key cryptography (AES-128bits) Alliance (2017) for both encryption/decryption and MAC operations. The specification defines two ways for a device to obtain the keys necessary to take part in a LoRaWAN network (this is called activation of the device):

- Over-The-Air Activation (OTAA)
- Activation By Personalization (ABP)

Put simply, OTAA refers to remote activation and ABP refers to manual activation, where keys are pre-configured in the device. In both versions, ABP is very similar (although the specific keys configured are different): the ED is connected, for example, via JTAG or USB connector and all the keys and related material ($DevEUI$, etc) are transferred to secure storage in the device (hence, attacks to this procedure are very limited). In OTAA, the ED asks permission to connect to the LoRaWAN network. This is achieved by successful transmission and verification of *join-request* and *join-accept* messages. The content of these messages differ in both versions and will be described below in more detail.

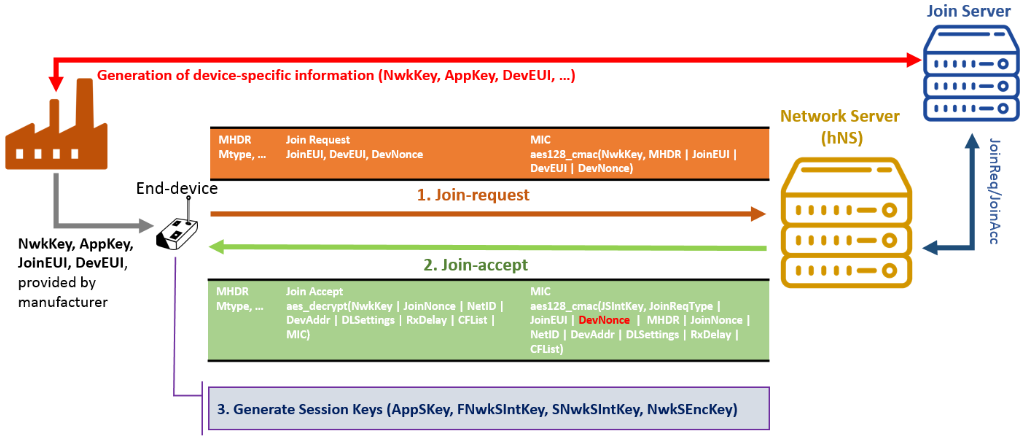


Figure 2: LoRaWAN v1.1 Over the Air Activation (OTAA) procedure

2.1.1. LoRaWAN v1.0

Figure 1 depicts the OTAA procedure for the version 1.0 of the LoRaWAN. First, the ED gathers the $AppEUI$ and the $AppKey$ from the network manager (this can happen through some manual configuration). Then, when the ED is deployed, it communicates with the NS via a gateway to initiate the *OTAA Join Procedure*. The session starts with the *join-request* message sent from ED to NS. The NS checks the integrity of the message and after validation, it forwards the *join-request* to the Application Server (AS) which checks the entry for this specific ED in the *Supported Devices List*, matching the $DevEUI$ of the ED to its associated $AppKey$. After a successful match, the AS responds with an $AppNonce$ to the NS. Then, the NS appends a $NETID$ and also some radio and configuration parameters, along with a Message Integrity Code (MIC) to send back to the ED in the *join-accept* message. The ED validates the MIC and then decrypts the message to obtain the $AppNonce$, $NETID$ and parameters. Finally, $AppNonce$ and $NETID$ are used to create session-long keys: $AppSKey$ and $NwkSKey$. These session keys are used for confidentiality and integrity of the messages exchanged afterwards.

2.1.2. LoRaWAN v1.1

In the architectural layout of LoRaWAN v1.1 networks, a new server called Join Server (JS) is introduced to manage the OTAA procedure. Furthermore, instead of a single NS, there are three NS roles introduced: home, forwarding and serving. The logic behind these modifications is to make roaming of the devices possible.

As in the case of v1.0, v1.1 also employs same two mechanisms for key distribution, namely ABP and OTAA. There is no change in the ABP procedure, however the OTAA is significantly changed in v1.1. Figure 2 depicts the OTAA procedure for version 1.1 of the LoRaWAN. Here, the unique identifier of the JS, the $JoinEUI$ and the $DevEUI$ (unique identifier of the ED) and both are pre-configured in the ED during fabrication. The ED also needs to be configured with the $NwkKey$ and the *AppKey* (this can happen also during fabrication). Then, when the ED is deployed, it communicates with the Join Server (JS) via NS, through a gateway to initiate the $OTAA\ Join$ *Procedure*. The session starts with the *join_request* message sent from the ED. The receiving NS checks the message and forwards the request to the JS which checks the entry for this specific ED in the *Supported Devices List*, matching the $DevEUI$ of the ED to its associated $NwkKey$ and *AppKey*. After a successful match, the JS responds with a $JoinNonce$. Then, the NS appends a $NETID$ and also some radio and configuration parameters, along with a Message Integrity Code (MIC) to send back to the ED in the *join_accept* message. ED validates the MIC and then decrypts the message to obtain the $JoinNonce$, $NETID$ and parameters.

To finalize the OTAA procedure, the $JoinNonce$, $JoinEUI$, $DevNonce$ and $NwkKey$ are used to create network session-long keys: $NwkSEncKey$, $FNwkSIntKey$, $SNwkSIntKey$. The $FNwkSIntKey$ - Forwarding Network Session Integrity Key - is used for the message integrity code (MIC) of uplink data messages. Whereas, the $SNwkSIntKey$ - called Serving Network Session Integrity Key - is used for the message integrity code (MIC) of downlink data messages. $NwkSEncKey$ and *AppSKey* keys (network and application) are used for confidentiality and integrity of the messages exchanged afterwards. Finally, $JoinNonce$, $JoinEUI$, $DevNonce$ and *AppKey* are used to create application session-long key:*AppSKey*, the session key shared between the ED and AS and used to encrypt/decrypt application layer payloads.

Readers who are more interested in details of LoRaWAN v1.1 may refer to Butun et al. (2018). There, a comprehensive comparison table is provided to enlighten readers about what changes are introduced with the new version, especially from the security point of view (new keys, nounces, frame counters, etc.).

2.2. Known Attacks to LoRaWAN

LoRaWAN was studied by many previous works which have investigated its security and proposed enhancements to the specification. Some of these enhancements were included in LoRaWAN v1.1, and, due to its recent release, there is no work dedicated to analyzing its security yet. In this subsection, we will review work on the previous version of LoRaWAN (v1.0) which might no longer be applicable to the latest version (v1.1) due to the changes introduced. This prior art is still useful to acknowledge the progress of the specification and to have an overview of previous attacks and improvements proposed. In Section 2.3, we provide a discussion about the limitations of Scyther and of the presented model in view of the vulnerabilities described in this section such that the reader can better grasp the security properties that can be derived from the model and its limitations.

The authors of Antipolis and Girard (2015) have focused on a problem with LoRaWANs key management methodology. In the version 1.0 of LoRaWAN, the NS is responsible for generating both session keys: the $NwkSKey$ and $AppSKey$. This is vulnerable to attacks since NS possesses the $AppSKey$, it can decrypt and read any message passing by. As a solution to this problem, authors proposed a new LoRaWAN network architecture in which PKI is employed as a trusted entity. Fortunately, this vulnerability (lack of root keys separation) was already addressed in the new version of LoRaWAN (v1.1) as the derivation of $NwkSKey$ and $AppSKey$ comes from different root keys. In our presented model we examine the communication between ED and NS, as the OTAA session is carried out by these two entities. Although the connections between AS, JS, and NS are not covered by our model, we assume insider server connections are less likely to be vulnerable to cyberattacks, as mentioned in the protocol standard Alliance (2017).

Another related work Kim and Song (2017), proposed an improved scheme using dual keys for ED activation to improve the separation of trust for the management of session keys. Eventually, this proposal somewhat is accepted and inherited by the latest version of LoRaWAN (v1.1), since a new root key ($NwkKey$) is introduced to generate the $NwkSKey$. With the inclusion of the new root key ($NwkKey$), the session keys of application and network sessions are generated separately (each session key is generated by its' own root key) during the OTAA activation phase of LoRaWAN v1.1.Similar to the previous paper of Antipolis and Girard (2015), this work tried to present an improvement to v1.0 by employing root key separation, which was already addressed in v1.1.

The $DevNonce$ required in LoRaWAN v1.0 is a random number created by the EDs. It is used to circumvent replay attacks during the key generation phase. Zulian has shown that with the $DevNonce$ generation system of LoRaWAN v1.0, after a certain period of time, the ED can be unavailable with a certain probability Zulian (2016). To tackle this problem, author proposed increasing the size of the $DevNonce$ field up-to 24-32 bits.

The same problem was also topic of another research by Tomasin et al. (2017). Authors stressed that, by using specific jamming techniques, the $DevNonce$ number pool can be finished in a short duration of time. Accordingly, after a while, NS will start to drop all of the *join-request* messages from that ED because the nonces it possess are simply used already. These issues are related to the $DevNonce$ randomization strength, which is not an investigated property by the automated security verification tools (i.e., Scyther). Since Scyther assumes perfect randomization technique over ideal cryptographic conditions, it could not identify this vulnerability in our model. Luckily, this issue has been addressed in LoRAWAN v1.1 as well.

Some of the LoRaWAN v1.0's security vulnerabilities and related remedies are discussed in Miller (2016). This work reported several vulnerabilities in the phases of key management, communications, and network connection.

Na et al. (2017) argued that the *join-request* message sent by the ED to the NS during the OTAA procedure is not encrypted and therefore vulnerable to replay attacks. They have even proposed a remedy to prevent this. However, authors have missed the point that, NS is keeping the list of used $DevNonce$s and automatically protects the network from bad ramifications of the replay attacks. This is a kind of replay attack, which has been reported by our model for v1.0. The new version of LoRAWAN tackles this vulnerability by sending back the received $DevNonce$ contained within the *join-accept* message.

Regardless of the version being used for LoRaWAN networks (either v1.0 or v1.1), owing to wireless communications technology, they are susceptible to not only inter-network interference but also jamming attacks. The threat for LoRa is not as serious as in the other narrow-band wireless technologies. Hence, the CSS modulation of the LoRa spreads the use of the communication channels to a wider band, the bad effects of these problems are somewhat solved. However, more complicated jamming attacks, such as a selective-jamming attack, cannot be detected easily and would result in the decrease of the network performance. Jamming attacks are related to the physical layer and Scyther can only check security issues within the logic of the protocol (cryptographically). Therefore, in our analysis, Scyther is not able to identify this kind of attack.

Sanchez-Iborra et al. (2018) *et al.*'s work is the most recent paper to address security issues of LoRaWAN v1.0 and to offer remediation by proposing a lightweight and authenticated key management approach. The proposed approach is based on the Ephemeral Diffie Hellman Over COSE (EDHOC) and defined as a convenient solution due to its flexibility in the update of session keys, its low computational cost and the limited message exchanges needed. The paper includes a comparative conceptual analysis by considering the overhead of possible implementations of rival security schemes for LoRaWAN v1.0. However, authors did not work with latest version of LoRaWAN (v1.1). Therefore, their work needs to be expanded and revisited, considering the significant security improvements included in v1.1.

2.3. Security Protocol Analysis and Scyther

Security protocol formal verification tools have received a great attention in the last few years. These tools had a role in improving some security protocols even after being adopted Dalal et al. (2010). Scyther by Cremers (2006), ProVerif by Blanchet et al. (2001), and Avispa by Armando et al. (2005) are some notable examples of formal verification tools, that, while having a similar objective, they vary in their coding and validation method, as mentioned in Dalal et al. (2010).

Dalal et al. (2010) also stresses that Scyther is be one of the most well-known tools for security validation by offering a graphical analysis to demonstrate security threats based on protocol models outlined using the Security Protocol Description Language (or SPDL programming language). Scyther evaluates the examined protocol against predefined security claims that are also included in the model and allows validating the protocol for either an unbounded or bounded number of sessions. It can also use a characterized role to analyze the protocol by performing a complete execution that demonstrates all traces of the protocol role.

The two pivotal claims in our evaluation are the non-injective synchronization claim (or $Nisynch$) as well as the non-injective agreement claim (or $Niagree$). Synchronization states that the exchanged messages are transferred exactly as set by the protocol description. However, agreement only cares about the final variable values after a successful completion between two communications parties regardless of what happens in between. Synchronization can be show to be stronger than agreement in the typical intruder model.

In other sense, according to Cremers et al. (2006); Lowe (1997), synchronized protocols are not vulnerable to replay, suppress-replay, and pre-play attacks, while agreeing protocols probably are.

It is worth mentioning that in the literature, Scyther verification tool has already been useful in analyzing the security vulnerabilities of some communication standards: It has shown by Dalal et al. (2010) that following protocols need improvements for their security standardizations: WiMAX, Extensible Authentication Protocol (EAP, which is a network access authentication framework), and ISO/IEC 9798 (entity authentication protocol).

Scyther assumes ideal/perfect cryptographic conditions with unbreakable encryption, in which the opponent can learn nothing from the encrypted message without the decryption key(s). In Scyther there is no difference between, for example, Data Encryption Standard (DES) and Advanced Encryption Standard (AES), as both are considered perfect symmetric key encryption ciphers. This presents one of the main Scyther limitations which is discussed in more depth by Yang et al. (2016b,a).

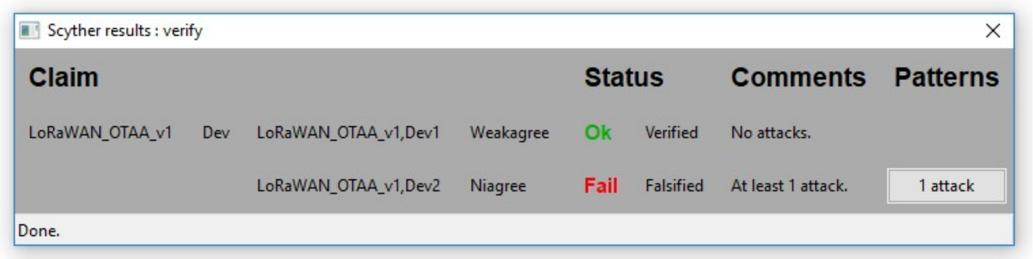
In addition, Scyther only deals with the logical part of the security protocols, in view of that, our model is not successful to detect some of the previous illustrated shortcomings in Section 2.2: (i) nonce randomization weaknesses, (ii) root keys separation in the derivation of $NwkSKey$ and $AppSKey$, and (iii) physical attacks in terms of radio jamming. On the contrary, the presented model successfully detected the replay attack vulnerability in v1.0, as the lack of $Nisynch$ (non-injective synchronization) directly leads to replay, suppress-replay, and/or pre-play attacks Cremers (2006). The current version of LoRAWAN covered this vulnerability by sending back the received $DevNonce$ to the ED. Scyther can only simulate cryptographic Hash functions, random nonce generation and symmetric/asymmetric key encryption. Other cryptographic functions are not directly supported by Scyther, such as: (i) key agreement over discrete logarithm problem (DLP), (ii) integer factorization problem (IFP), (iii) Chinese reminder theorem (CRT), (iv) time stamping/synchronization, and (v) exclusive-or. To overcome this constraint, analyzers/investigators need to manipulate the well-defined and supported properties to simulate the missing functions. This kind of limitations have been addressed by some other tools like ProVerif Küsters and Truderung (2009).

Listing 1: Pseudo-code for LoRaWAN-OTAA-v1.0 key agreement procedure

```
Color code:
red:operation, green:operand type,  orange:actors, blue: variables, magenta: constants

declaration of LoRaWAN OTAA v1.0 protocol to be comprised of Server and ED
begin
     declaration of ED role
     begin
         declaration of DevNonce,MHDRDev in type Nonce
         declaration of MHDRSrv,SrvNonce,NetID,DevAddr,DLSettings,RxDelay,CFList in type Nonce
         declaration of pad01, pad02,pad16 in type Padding

         send from ED to Server the message (MHDRDev, EDe, Server, DevNonce) and the HMAC of the
      message encrypted with the AppKey
         receive from Server at ED the message (MHDRSrv) and the secret
      message (SrvNonce, NetID, DevAddr, DLSettings, RxDelay, CFList) encrypted with the AppKey
         decrypt the secret message (SrvNonce, NetID, DevAddr, DLSettings, RxDelay, CFList) with the AppKey
         calculate AppSKey by encrypting the (pad01, SrvNonce, DevNonce, NetID, pad16) with the AppKey
         calculate NwkSKey by encrypting the (pad02, SrvNonce, DevNonce, NetID, pad16) with the AppKey

         check whether both parties (ED and Server) have the same value of DevNonce
         check aliveleness of the ED
         check the minimum agreement between the partees according to the ED
         check the validity of the non-injective agreement according to the ED
         check the validity of the non-injective synchronization according to ED
         check the validity of the secrecy of AppSKey according to the ED
         check the validity of the secrecy of NwkSKey according to the ED
     end

     declaration of Server role
     begin
         declare SrvNonce,MHDRSrv,NetID,DevAddr,DLSettings,RxDelay,CFList,NonceList in type Nonce
         declare DevNonce,MHDRDev in type Nonce

         receive from ED at Server the message (MHDRDev, ED, Server, DevNonce) and the HMAC of the
      message encrypted with the AppKey
         check whether DevNonce do not match the NonceList
         send from Server to ED the message (MHDRSrv) and the secret
      message (SrvNonce, NetID, DevAddr, DLSettings, RxDelay, CFList) encrypted with the AppKey
         update the NonceList by adding DevNonce

         check whether both parties (ED and Server) have the same value of SrvNonce
         check aliveleness of the Server
         check the minimum agreement between the partees according to the Server
         check the validity of the non-injective agreement according to the Server
         check the validity of the non-injective synchronization according to Server
         check the validity of the secrecy of AppSKey according to the Server
         check the validity of the secrecy of NwkSKey according to the Server
     end
end
```

3. Security Analysis of LoRaWAN

As in every kind of security implementation, the security of LoRaWAN includes several dimensions, such as: protocol issues, user behavior, implementation aspects, weaknesses in the cryptography algorithms employed. In this section, we will focus in the scope on automated security protocol verification tools and develop a model to verify the security of LoRaWAN, particularly the OTAA procedure.



Claim				Status		Comments	Patterns
LoRaWAN_OTAA_v1	Dev	LoRaWAN_OTAA_v1,Dev1	Weakagree	Ok	Verified	No attacks.	
		LoRaWAN_OTAA_v1,Dev2	Niagree	Fail	Falsified	At least 1 attack.	1 attack

Done.

Figure 3: LoRaWAN v1.0 OTAA Scyther validation results: a sample output

3.1. LoRaWAN v1.0

In the LoRaWAN v1.0 OTAA, the *ED* sends a *join-request* message to the *NS* to authenticate itself (to validate this request, the intervention of the *AS* is needed, but for the purposes of our model, we treat the Network and Application Servers as one entity). With a successful validation, the server answers the *ED* with a unique response, named a *join-accept* message, that carries shared key parameters, such as *AppNonce* and *NetID*. In order to facilitate the understanding of the model, we present an English-readable pseudo-code for the model in Listing 1. Listing 3 in the Appendix A section provides the full SPDL code of our Scyther model for LoRaWAN v1.0 OTAA session.

One can observe the two roles modeled: *ED* and *Server*. The *ED* computes a key, *JSIntKey* and a MIC that are sent to the *NS* from whom we then expect a reply (the *join-accept*). We can also see (Listing 1) several security claims checked by the model. Our model of the *Server* is similar to the *ED*, but the *Server* receives the *join-request* and then replies to it. The *Server* also checks if the *DevNonce* was not previously used.

3.1.1. Security Verification Results

After execution of the model, the results generated by Scyther can be obtained (in an output windows such as in Figure 3). For example, Figure 3 shows a sample output of Scyther results for the analysis of LoRaWAN v1.0 OTAA. Accordingly, Scyther provided results for each claim we have created by showing the security implications related to them. In the cases of “Fail, the possible attack scenarios detected by Scyther can be manually inspected. A click-able button in the “Patterns column (button with the name “1 attack in Figure 3) opens a new window in which a possible attack scenario is plotted. For the sake of simplicity, this attack scenario is not shown here. More interested readers can run the Scyther code provided in Listing 3 to observe the attack scenarios in detail. For convenience, we have summarized all the results of the Scyther analysis in Table 2. In this table, $N.A.$ refers to "Not Applicable". Hence we have merged the results from both versions (v1.0 and v1.1) in a single table, some claims are not valid for the specific version while they are valid for the other. All those claims are indicated with $N.A.$ label.

The results summarized in Table 2 show that two security claims - $Nisynch$ and $Niagree$ - are not satisfied, and at least one attack can be performed. In practical terms, what this means is that LoRaWAN v1.0 OTAA does not provide strong ties between the two communicating parties (i.e., the ED and the NS/AS). In other sense, there is a weak relation between the *join-request* and *join-accept* messages for the same ED; the two communicating parties cannot be assured that they possess the same keying credentials, in the sense that if there are multiple join requests, the replies do not have information about which request they relate to.

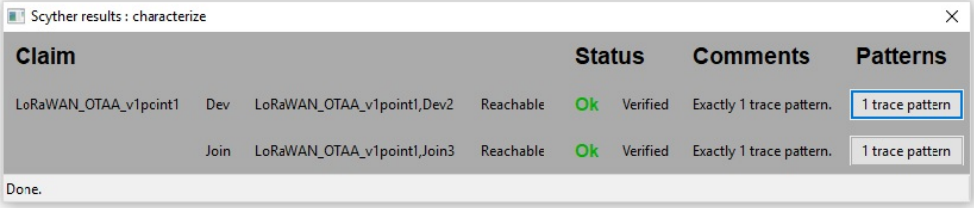
The consequences of missing agreement and synchronization properties between the communicating parties can be, for example, when the system loses the *send_1* request of a first join attempt, it will count the future *send_1* request of the third run instead. This, in turn, will lead the two parties to agree on dissimilar keying materials as they will have different nonce values.

Other claims / checks such as; i. *SKR* refers to the secrecy of certain attributes, preferred to be utilized for session keys, ii. *Alive* assures the liveliness of all partners. iii. *Weakagree* tends to a weak agreement, in which the communication partners need to assure that they are actually communicating with each other to prevent an attacker from impersonating one of them. More details can be found in Lowe (1997).

3.1.2. Discussion

Scyther results are based on an abstract model of the protocol, but it is interesting to verify that similar attack scenarios have been disclosed previously by Tomasin et al. (2017); Na et al. (2017), where authors reported jamming and replay attacks to the LoRaWAN join procedure. To address this problem, a strong tie between the two parties must be established. That is, the ED needs to be confident that the NS/AS is obtaining the same $DevNonce$ that is sent over the *join-request* message.

To address this issue, the server has to include the received $DevNonce$ to its *join-accept* message and send it back to the device in a ciphered format. This could be solved as follows; In the join request message, MIC is

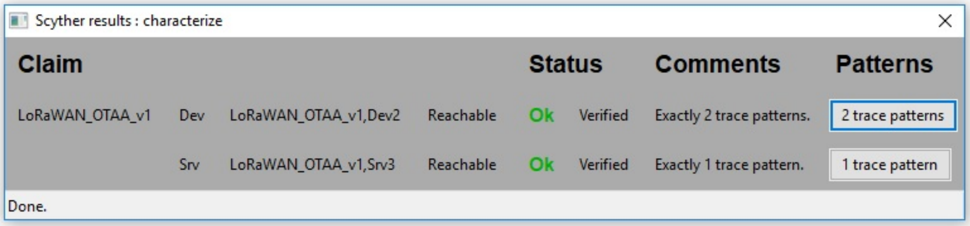


Scyther results : characterize

Claim				Status		Comments	Patterns
LoRaWAN_OTAA_v1point1	Dev	LoRaWAN_OTAA_v1point1,Dev2	Reachable	Ok	Verified	Exactly 1 trace pattern.	1 trace pattern
	Join	LoRaWAN_OTAA_v1point1,Join3	Reachable	Ok	Verified	Exactly 1 trace pattern.	1 trace pattern

Done.

Figure 4: LoRaWAN v1.1 Scyther characterize role



Scyther results : characterize

Claim				Status		Comments	Patterns
LoRaWAN_OTAA_v1	Dev	LoRaWAN_OTAA_v1,Dev2	Reachable	Ok	Verified	Exactly 2 trace patterns.	2 trace patterns
	Srv	LoRaWAN_OTAA_v1,Srv3	Reachable	Ok	Verified	Exactly 1 trace pattern.	1 trace pattern

Done.

Figure 5: LoRaWAN v1.0 Scyther characterize role

replaced by an AES Encryption to help the server to extract a $DevNonce$ image. Subsequently, in the *join-accept* response, the server integrates an $XOR(DevNonce, SrvNonce)$ to allow the ED to check that the server is really obtaining the corresponding $DevNonce$. In the next subsection, we will see how LoRaWAN v1.1 remedies this problem.

3.2. LoRaWAN v1.1

In this subsection, we examine LoRaWAN v1.1. In an earlier section (Section 2.1.2) the key agreement process (OTAA) of LoRaWAN v1.1 was shown in details (for an illustration, see Figure 2). Again, to analyze security of this newer version, we start by modeling LoRaWAN v1.1 OTAA using Scyther.

3.2.1. Security Verification Results

The Scyther validation shows that all claims are verified and with no attacks including the two security claims $Nisynch$ and $Niagree$, as presented in Table 2. For the reader's convenience, Listing 2 presents the pseudo code of the Scyther model. Whereas Listing 4 presents the full SPDL code of the

Listing 2: Pseudo-code for LoRaWAN-OTAA-v1.1 key agreement procedure

```
Color code:
red:operation, green:operand type,  orange:actors, blue: variables, magenta: constants

declaration of LoRaWAN OTAA v1.1 protocol to be comprised of ED and Join-Server
begin
    declaration of the role of ED
    begin
        declaration of DevNonce and MHDRDev in type Nonce
        declaration of MHDRSrv, JoinNonce, NetID, DevAddr in type Nonce
        declaration of DLSettings, RxDelay, CFList, JoinReqType in type Nonce
        declaration of JSIntKey, MIC, AppSKey, JSEncKey in type Key
        declaration of FNwkSIntKey, SNwkSIntKey,NwkSEncKey in type Key
        declaration of pad01,pad02,pad03,pad04,pad05,pad06,pad16 in type Padding

        calculate JSIntKey as (pad06,ED,pad16) encrypted with the secret-key between ED and Join-Server
        calculate MIC as (JoinReqType,Join-Server,DevNonce, MHDRSrv, JoinNonce,NetID,DevAddr,DLSettings,RxDelay,CFList) encrypted with the JSIntKey
        send from ED to Join-Server the message (MHDRDev, ED, Join-Server, DevNonce) and the HMAC of the message encrypted with the secret-key between ED and Join-Server
        receive from Join-Server at ED the message (MHDRSrv, MIC, (JoinNonce,NetID,DevAddr,DLSettings,RxDelay,CFList,MIC) encrypted via decrypt option with the secret-key between ED and Join-Server)
        calculate FNwkSIntKey as (pad01,JoinNonce,Join-Server,DevNonce,pad16) encrypted with the secret-key between ED and Join-Server
        calculate SNwkSIntKey as (pad03,JoinNonce,Join-Server,DevNonce,pad16) encrypted with the secret-key between ED and Join-Server
        calculate NwkSEncKey as (pad04,JoinNonce,Join-Server,DevNonce,pad16) encrypted with the secret-key between ED and Join-Server

        calculate AppSKey as (pad02,JoinNonce,Join-Server,DevNonce,pad16) encrypted with the public-key of the Application Server (Appkey)
        calculate JSEncKey as (pad05,ED,pad16) encrypted with the secret-key between ED and Join-Server

        check whether both parties (ED and Join-Server) have the same value of DevNonce
        check aliveleness of the ED
        check the minimum agreement between the partees according to the ED
        check the validity of the non-injective agreement according to the ED
        check the validity of the non-injective synchronization according to ED
        check the validity of the secrecy of FNwkSIntKey according to the ED
        check the validity of the secrecy of SNwkSIntKey according to the ED
        check the validity of the secrecy of NwkSEncKey according to the ED
        check the validity of the secrecy of AppSKey according to the ED
        check the validity of the secrecy of JSEncKey according to the ED
        check the validity of the secrecy of JSIntKey according to the ED
    end

    declaration of the role of Join-Server
    begin
        declaration of JoinNonce, MHDRSrv, NetID, DevAddr, DLSettings in type Nonce
        declaration of RxDelay, CFList,NonceList,JoinReqType, in type Nonce
        declaration of DevNonce, MHDRDev in type Nonce

        receive from ED at Join-Server the message (MHDRDev, ED, Join-Server, DevNonce) and the HMAC of the message encrypted with the secret-key between ED and Join-Server
        send from Join-Server at ED the message (MHDRSrv, MIC, (JoinNonce,NetID,DevAddr,DLSettings,RxDelay,CFList,MIC) encrypted via decrypt option with the secret-key between ED and Join-Server)

        check whether DevNonce do not match the NonceList
        update the NonceList by adding DevNonce
        check whether both parties (Join-Server and ED) have the same value of JoinNonce
        check aliveleness of the Join-Server
        check the minimum agreement between the partees according to the Join-Server
        check the validity of the non-injective agreement according to the Join-Server
        check the validity of the non-injective synchronization according to Join-Server
        check the validity of the secrecy of FNwkSIntKey according to the Join-Server
        check the validity of the secrecy of SNwkSIntKey according to the Join-Server
        check the validity of the secrecy of NwkSEncKey according to the Join-Server
        check the validity of the secrecy of AppSKey according to the Join-Server
        check the validity of the secrecy of JSEncKey according to the Join-Server
        check the validity of the secrecy of JSIntKey according to the Join-Server
    end
end
```

Table 2: LoRaWAN v1.0 and v1.1 OTAA Scyther validation results

Claim	**LoRaWAN v1.0**		**LoRaWAN v1.1**	
	Status	**Attack patterns**	**Status**	**Attack patterns**
Reference: End Device				
Alive	Ok	No attacks	Ok	No attacks
Weakagree	Ok	No attacks	Ok	No attacks
Niagree	**Fail**	**1+ attacks**	Ok	No attacks
Nisynch	**Fail**	**1+ attacks**	Ok	No attacks
SKR{AppSKey}	Ok	No attacks	Ok	No attacks
SKR{NwkSKey}	Ok	No attacks	N.A.	N.A.
SKR{SNwkSIntKey}	N.A.	N.A.	Ok	No attacks
SKR{NwkSEncKey}	N.A.	N.A.	Ok	No attacks
SKR{JSEncKey}	N.A.	N.A.	Ok	No attacks
SKR{JSIntKey}	N.A.	N.A.	Ok	No attacks
Reference: Server				
Alive	Ok	No attacks	Ok	No attacks
Weakagree	Ok	No attacks	Ok	No attacks
Niagree	Ok	No attacks	Ok	No attacks
Nisynch	Ok	No attacks	Ok	No attacks
SKR{AppSKey}	Ok	No attacks	Ok	No attacks
SKR{NwkSKey}	Ok	No attacks	N.A.	N.A.
SKR{SNwkSIntKey}	N.A.	N.A.	Ok	No attacks
SKR{NwkSEncKey}	N.A.	N.A.	Ok	No attacks
SKR{JSEncKey}	N.A.	N.A.	Ok	No attacks
SKR{JSIntKey}	N.A.	N.A.	Ok	No attacks

Scyther model for LoRaWAN v1.1 OTAA procedure. In our Scyther model for LoRaWAN v1.1 (Listing 2), one can observe that the message sent from *Server* to *ED*, (*send* in line 45 and *receive* in line 18) includes a MIC that is computed with the $DevNonce$. As we can observe, a small change in the protocol addresses the weaknesses previously found for LoRaWAN v1.0. Including the $DevNonce$ in the MIC of the *join-accept* message results in an unequivocal correspondence between pairs of join request/accept messages and ensures that both communicating parties end up having the same keying materials.

3.2.2. Discussion

Executing Scyther validation tool against the defined claims is still not sufficient to give a precise examination of the inspected protocol. As protocol designers, who are trying to protect their protocol against illegal access, may, *accidentally*, block the protocol for the authorized access as well. The Scyther characterize role carries out the responsibility of checking the reachability of each partner in the network to assure that the protocol can be run smoothly and efficiently between its legitimate users during the execution phase. We examined the characterize role against LoRaWAN v1.1 to extract a related window, shown in Figure 4, to prove that the communications' partners are reachable to each other over a single (authentic) trace pattern. Usually multiple traces reflect potential vulnerabilities. The characterize role for LoRaWAN v1.0, shown in Figure 5, states that the *Dev* entity can be reached over (two) different traces. One of these traces covers the legitimate access and the other trace presents a related weakness, for more details please check Section 3.1

While our model does not report issues with version 1.1 of LoRaWAN, this does not mean that the protocol is free from security vulnerabilities, but if does give strong indications, particularly in regards to the claims made in the model.

4. Open Security Challenges with LoRaWAN v1.1

The Scyther models presented allow to derive some important security properties. There are however still some concerns and open challenges for further research. In this section, we highlight some important open security challenges.

4.1. Cryptographic primitives

As discussed earlier, one major limitation of automatic protocol verification is that it generally considers the cryptographic primitives to be ideal. In practice, there might be weaknesses in the cryptographic primitives that would impact on the security of the protocol. As an example, researchers have previously described some fundamental flaws in AES using the electronic codebook (ECB) mode Rogaway (2011), used to encrypt the *join-accept* message of LoRaWAN v1.1.

4.2. Key Preloading

In the key agreement context, the (joint) *key-control property* prevents any party in the network from selecting a predefined value for the shared session key. Doing this stops one party from having any kind of benefits over the other party Mitchell et al. (1998). The preloading of the root keys in LoRaWAN v1.1 (*NwkKey* and *AppKey*) into the ED violates this expected key-control property. The main advantage of the key-control property is to guarantee the independence in the key agreement process for the concerned parties Eldefrawy et al. (2011). In addition to that, key preloading requires extra resources in terms of separate and secure means for the loading process.

4.3. Infrastructure Trust

Yang (2017) presented many security vulnerabilities of LoRaWAN v1.0. Especially, the work mentioned a specific version of man-in-the-middle(MITM) attack called bit-flipping attack, in which an adversary (or a rogue NS) changes the content of the messages in between NS and AS. This attack is still valid for v1.1 as mentioned in the specification document, Alliance (2017): “Application payloads are end-to-end encrypted between the ED and the AS, but they are integrity protected only in a hop-by-hop fashion; one hop between the ED and the NS, and the other hop between the NS and the AS. That means, a malicious NS may be able to alter the content of the data messages in transit, which may even help the NS to infer some information about the data by observing the reaction of the application end-points to the altered data. ” Therefore, as stressed by the specification document, NSs are considered as trusted servers by default. However, entities are recommended to use additional end-to-end security solutions if they are wishing to implement end-to-end confidentiality and integrity protection against MITM attacks.

4.4. Roaming

Roaming support is one of the major aspects introduced in LoRaWAN v1.1. Our model does not include security aspects related to roaming operations, which is left as a future work. However, here we will briefly state and summarize two related considerations: (i) As mentioned previously, v1.1 of LoRaWAN is susceptible to bit-flipping attacks happening in between servers as much as the v1.0. The inclusion of handover-roaming in v1.1 makes the situation worse. As discussed in Dönmez and Nigussie (2018), handover-roaming enables more possibilities for a MITM attack, as the unprotected *FRMPayload*'s are first transported from the sNS (serving-NS) to the hNS (homing-NS), and from there to the AS; (ii) As stressed by Dönmez and Nigussie (2018), handover-roaming can cause a fall-back when the back-end (sNS) that serves the roaming ED runs an older version of LoRaWAN, i.e. v1.0. On contrary to this thought, handover-roaming is itself a v1.1 feature and is not presented in v1.0. Henceforth, handover-roaming from LoRaWAN v1.1 network into a v1.0 network is simply not allowed. Handover-roaming depends on the trust of only the network session keys. As far as the network operators entrust the network root keys delivered to them by the other operators they have roaming agreements with, handover-roaming should not introduce extra security implications in regards to join procedure commissioning.

5. Conclusion and Future Work

LoRaWAN, with its very desirable features such as low-cost and long-range communications, is increasingly being considered as an option to deploy IoT networks. In this article, using the Scyther verification tool, we show that LoRaWAN's release v1.0 suffers from a lack of synchronization between the communicating parties, which in its turn, makes it vulnerable to a known family of attacks: replay attacks. Interestingly, the vulnerabilities found using an abstract model of the protocol are practical, as previously reported independently Tomasin et al. (2017); Na et al. (2017). On the other hand, the latest version of LoRaWAN (v1.1) has passed all the security claims/checks of our model. However, due to the limitations of the model, it is not possible to discover all the potential vulnerabilities of a protocol using tools like Scyther. In fact, we also have discussed some security challenges of LoRaWAN 1.1 that need of further discussion.

These results are relevant several ways: (i) LoRaWAN v1.0 is still widely used and our discussion shows that it possible to address the weaknesses in this v1.0 of the protocol whereas an upgrade to v1.1 might require changes in the infrastructure and more time; (ii) they show how a formal model can successfully find practical protocol weaknesses (iii) they provide a discussion on the security of the protocol and on the usefulness and limitations of automated protocol verification.

Scyther acts as a microscope to security protocols; it allows examining their security properties with great detail. Not only that, but also it can help designing and checking solutions to security issues found. We note that the SPDL code presented in this work provides realistic models for LoRaWAN v1.0 and v1.1, and we believe that this work lays a foundation to check the security of LoRaWAN, including adding other features of the protocol to the model as well as modifying them to model future updates or releases of the protocol.

Acknowledgement

The authors would like to thank the anonymous reviewers and our Shepherd (Doctor Rodrigo Román Castro) for their valuable comments and feedback on the several versions of our manuscript. This work was supported by grant 20150367 of the Swedish Knowledge Foundation, by grant 20201010 (SMART Project) of the European Regional Fund, and the Portuguese funding institution FCT - Fundação para a Ciência e a Tecnologia, under the sabbatical leave fellowship SFRH/BSAB/128459/2017.

Appendix A. Scyther SPDL code

The Appendix section consists of following:

- The Listing 3 provides the Scyther code for the OTAA procedure of LoRAWAN v1.0
- The Listing 4 provides the Scyther code for the OTAA procedure of LoRAWAN v1.1

Listing 3: Scyther SPDL code for LoRaWAN-OTAA-v1.0

```
// The protocol is running between End Device (Dev) and NS/AS (Srv)
// The predefined shared key AppKey between Dev and Srv is k(Dev,Srv)
// dec models a decryption function that is invertible by an encryption function (enc)
// Declaration of padding strings (pad01, pad02, ...) omitted

protocol LoRaWAN-OTAA-v1(Dev,Srv)
{
  role Dev {
    fresh DevNonce: Nonce;
    fresh MHDRDev: Nonce;
    var MHDRSrv: Nonce;
    var SrvNonce: Nonce;
    var NetID: Nonce;
    var DevAddr: Nonce;
    var DLSettings: Nonce;
    var RxDelay: Nonce;
    var CFList: Nonce;
    send_1(Dev,Srv,(MHDRDev, Dev,Srv,DevNonce),{MHDRDev,Dev,Srv,DevNonce}k(Dev,Srv));
    recv_2(Srv,Dev,(MHDRSrv), {{SrvNonce,NetID,DevAddr,DLSettings,RxDelay,CFList}dec}k(Dev,Srv),
    {SrvNonce,MHDRSrv,NetID,DevAddr,DLSettings,RxDelay,CFList}k(Dev,Srv));
    macro AppSKey={pad01,SrvNonce,DevNonce,NetID,pad16}k(Dev,Srv);
    macro NwkSKey={pad02,SrvNonce,DevNonce,NetID,pad16}k(Dev,Srv);
    claim(Dev,Running,Srv,DevNonce); //checks that Dev agrees with Srv on DevNonce
    claim(Dev,Alive); //assures the Aliveness of Dev
    claim(Dev,Weakagree); //minimum agreement check between partners according to Dev
    claim(Dev,Niagree); //validates the non-injective agreement according to Dev
    claim(Dev,Nisynch); //validates the non-injective synchronization according to Dev
    claim (Dev,SKR,AppSKey); //validate the secrecy of AppSKey according to Dev
    claim (Dev,SKR,NwkSKey); //validate the secrecy of NwkSKey according to Dev
  }
  role Srv {
    fresh SrvNonce:Nonce;
    fresh MHDRSrv:Nonce;
    fresh NetID:Nonce;
    fresh DevAddr:Nonce;
    fresh DLSettings:Nonce;
    fresh RxDelay:Nonce;
    fresh CFList:Nonce;
    fresh NonceList:Nonce;
    var DevNonce:Nonce;
    var MHDRDev:Nonce;
    recv_1 (Dev,Srv,(MHDRDev,Dev,Srv,DevNonce), {MHDRDev,Dev,Srv,DevNonce }k(Dev,Srv));
    not match (DevNonce, NonceList);
    send_2 (Srv,Dev,(MHDRSrv ),{{SrvNonce,NetID,DevAddr,DLSettings,RxDelay,CFList}dec}k(Dev,Srv),
    { SrvNonce,MHDRSrv,NetID,DevAddr,DLSettings,RxDelay,CFList}k(Dev,Srv));
    macro NonceList = (NonceList, DevNonce);
    claim(Srv,Running,Dev,SrvNonce); //checks that Srv agrees with Dev on SrvNonce
    claim(Srv,Alive); //assures the Aliveness of Srv
    claim(Srv,Weakagree); //minimum agreement check between partners according to Srv
    claim(Srv,Niagree); //validates the non-injective agreement according to Srv
    claim(Srv,Nisynch); //validates the non-injective synchronization according to Srv
    claim (Srv,SKR,AppSKey); //validate the secrecy of AppSKey according to Srv
    claim (Srv,SKR,NwkSKey); //validate the secrecy of NwkSKey according to Srv
  }
}
```

Listing 4: Scyther SPDL code for LoRaWAN-OTAA-v1.1

```
// The protocol is running between End Device (Dev) and Network Server/Join Server (Join).
// The predefined shared key (NwkKey) between End Device and Server is k(Dev,Join).
// dec models a decryption function that is invertible by an encryption function (enc)
// Declaration of padding strings (pad01, pad02, ...) omitted
// Declaration of Appkey and NonceList as secrets omitted
protocol LoRaWAN-OTAA-v1point1 (Dev,Join)
{ role Dev {
    fresh DevNonce: Nonce;
    fresh MHDRDev: Nonce;
    var MHDRSrv: Nonce;
    var JoinNonce: Nonce;
    var NetID: Nonce;
    var DevAddr: Nonce;
    var DLSettings: Nonce;
    var RxDelay: Nonce;
    var CFList: Nonce;
    var JoinReqType: Nonce;
    macro JSIntKey={pad06,Dev,pad16 }k(Dev,Join);
    macro MIC={JoinReqType,Join,DevNonce,MHDRSrv,JoinNonce,NetID,DevAddr,DLSettings,RxDelay,CFList}JSIntKey;
    send_1(Dev,Join,(MHDRDev,Dev,Join,DevNonce),{MHDRDev,Dev,Join,DevNonce}k(Dev,Join));
    recv_2 (Join,Dev, (MHDRSrv),{{JoinNonce,NetID,DevAddr,DLSettings,RxDelay,CFList,MIC}dec} k(Dev,Join),MIC);
    macro FNwkSIntKey={pad01,JoinNonce,Join,DevNonce,pad16}k(Dev,Join);
    macro SNwkSIntKey={pad03,JoinNonce,Join,DevNonce,pad16}k(Dev,Join);
    macro NwkSEncKey={pad04,JoinNonce,Join,DevNonce,pad16}k(Dev,Join);
    macro AppSKey={pad02,JoinNonce,Join,DevNonce, pad16 }Appkey;
    macro JSEncKey={pad05,Dev,pad16}k(Dev,Join);
    claim(Dev,Running,Join,DevNonce); //checks that Dev agrees with Join on SrvNonce
    claim(Dev,Alive); //assures the Aliveness of Dev
    claim(Dev,Weakagree); //minimum agreement check between partners according to Dev
    claim(Dev,Niagree); //validates the non-injective agreement according to Dev
    claim(Dev,Nisynch); //validates the non-injective synchronization according to Dev
    claim (Dev,SKR,FNwkSIntKey); //validates the secrecy of FNwkSIntKey according to Dev
    claim (Dev,SKR,SNwkSIntKey); //validates the secrecy of SNwkSIntKey according to Dev
    claim (Dev,SKR,NwkSEncKey); //validates the secrecy of NwkSEncKey according to Dev
    claim (Dev,SKR,AppSKey); //validates the secrecy of AppSKey according to Dev
    claim (Dev,SKR,JSEncKey); //validates the secrecy of JSEncKey according to Dev
    claim (Dev,SKR,JSIntKey); //validates the secrecy of JSIntKey according to Dev
  } role Join {
    fresh JoinNonce: Nonce;
    fresh MHDRSrv: Nonce;
    fresh NetID: Nonce;
    fresh DevAddr: Nonce;
    fresh DLSettings: Nonce;
    fresh RxDelay: Nonce;
    fresh CFList: Nonce;
    fresh NonceList: Nonce;
    fresh JoinReqType: Nonce;
    var DevNonce: Nonce;
    var MHDRDev: Nonce;
    recv_1 (Dev,Join,(MHDRDev,Dev,Join,DevNonce),{MHDRDev,Dev,Join,DevNonce}k(Dev,Join));
    send_2 (Join,Dev,(MHDRSrv),{{JoinNonce,NetID,DevAddr,DLSettings,RxDelay,CFList,MIC}dec}k(Dev,Join),MIC);
    not match (DevNonce, NonceList);
    macro NonceList=(NonceList, DevNonce);
    claim(Join,Running,Dev,JoinNonce); //checks that Join agrees with Dev on JoinNonce
    claim(Join,Alive); //assures the Aliveness of Join
    claim(Join,Weakagree); //minimum agreement check between partners according to Join
    claim(Join,Niagree); //validates the non-injective agreement according to Join
    claim(Join,Nisynch); //validates the non-injective synchronization according to Join
    claim(Join, SKR, FNwkSIntKey); //validates the secrecy of FNwkSIntKey according to Join
    claim(Join, SKR, SNwkSIntKey); //validates the secrecy of SNwkSIntKey according to Join
    claim(Join, SKR, NwkSEncKey); //validates the secrecy of NwkSEncKey according to Join
    claim(Join, SKR, AppSKey); //validates the secrecy of AppSKey according to Join
    claim(Join, SKR, JSEncKey); //validates the secrecy of JSEncKey according to Join
    claim(Join, SKR, JSIntKey); //validates the secrecy of JSIntKey according to Join
  }
}
```

References

Alliance, L., 2017. LoRaWAN 1.1 Specification.
URL online:http://lora-alliance.org/lorawan-for-developers

Antipolis, S., Girard, P., 2015. Low Power Wide Area Networks security. white paper by Gemalto Inc.

Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuéllar, J., Drielsma, P. H., Héam, P.-C., Kouchnarenko, O., Mantovani, J., et al., 2005. The avispa tool for the automated validation of internet security protocols and applications. In: International conference on computer aided verification. Springer, pp. 281–285.

Blanchet, B., et al., 2001. An efficient cryptographic protocol verifier based on prolog rules. In: csfw. Vol. 1. pp. 82–96.

Butun, I., Pereira, N., Gidlund, M., 2018. Analysis of lorawan v1.1 security: Research paper. In: Proceedings of the 4th ACM MobiHoc Workshop on Experiences with the Design and Implementation of Smart Objects. SMARTOBJECTS '18. ACM, New York, NY, USA, pp. 5:1–5:6.

Cremers, C. J., Mauw, S., de Vink, E. P., 2006. Injective synchronisation: an extension of the authentication hierarchy. Theoretical Computer Science 367 (1-2), 139–161.

Cremers, C. J. F., 2006. Scyther: Semantics and verification of security protocols. Ph.D. thesis, Eindhoven University of Technology Eindhoven, Netherlands.

Dalal, N., Shah, J., Hisaria, K., Jinwala, D., 2010. A comparative analysis of tools for verification of security protocols. International Journal of Communications, Network and System Sciences 3 (10), 779.

Delbruel, S., Small, N., Hughes, D., 2017. Flip: Federation support for long range low power internet of things protocols. arXiv preprint arXiv:1712.08221.

Dönmez, T. C., Nigussie, E., 2018. Security of join procedure and its delegation in lorawan v1. 1. Procedia Computer Science 134, 204–211.

Eldefrawy, M. H., Khan, M. K., Alghathbar, K., 2011. Dynamic password based remote user authentication without time stamping.

Kim, J., Song, J., 2017. A dual key-based activation scheme for secure lorawan. Wireless Communications and Mobile Computing 2017.

Küsters, R., Truderung, T., 2009. Using proverif to analyze protocols with diffie-hellman exponentiation. In: Computer Security Foundations Symposium, 2009. CSF'09. 22nd IEEE. IEEE, pp. 157–171.

Lowe, G., 1997. A hierarchy of authentication specifications. In: Computer security foundations workshop, 1997. Proceedings., 10th. IEEE, pp. 31–43.

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

Mitchell, C. J., Ward, M., Wilson, P., 1998. Key control in key agreement protocols. Electronics Letters 34 (10), 980–981.

Na, S., Hwang, D., Shin, W., Kim, K.-H., 2017. Scenario and countermeasure for replay attack using join request messages in lorawan. In: Information Networking (ICOIN), 2017 International Conference on. IEEE, pp. 718–720.

Rogaway, P., 2011. Evaluation of some blockcipher modes of operation. Cryptography Research and Evaluation Committees (CRYPTREC) for the Government of Japan.

Sanchez-Iborra, R., Sánchez-Gómez, J., Pérez, S., Fernández, P. J., Santa, J., Hernández-Ramos, J. L., Skarmeta, A. F., 2018. Enhancing lorawan security through a lightweight and authenticated key management approach. Sensors (Basel, Switzerland) 18 (6).

Tomasin, S., Zulian, S., Vangelista, L., 2017. Security analysis of lorawan join procedure for internet of things networks. In: Wireless Communications and Networking Conference Workshops (WCNCW), 2017 IEEE. IEEE, pp. 1–6.

Vangelista, L., Zanella, A., Zorzi, M., 2015. Long-range iot technologies: The dawn of lora. In: Future Access Enablers of Ubiquitous and Intelligent Infrastructures. Springer, pp. 51–58.

Yang, H., Oleshchuk, V. A., Prinz, A., 2016a. Verifying group authentication protocols by scyther. JoWUA 7 (2), 3–19.

Yang, H., Prinz, A., Oleshchuk, V., 2016b. Formal analysis and model checking of a group authentication protocol by scyther. In: Parallel, Distributed, and Network-Based Processing (PDP), 2016 24th Euromicro International Conference on. IEEE, pp. 553–557.

Yang, X., 2017. Lorawan: Vulnerability analysis and practical exploitation. M.Sc. thesis, Delft University of Technology.

Zulian, S., 2016. Security threat analysis and countermeasures for lorawan join procedure. M.Sc. thesis, Universit‘a degli Studi di Padova.