Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 134 (2018) 204–211

Procedia Computer Science

www.elsevier.com/locate/procedia

The 13th International Conference on Future Networks and Communications (FNC 2018)

Security of Join Procedure and its Delegation in LoRaWAN v1.1

Tahsin C. M. Dönmez[a,*], Ethiopia Nigussie[a]

[a]*Department of Future Technologies, University of Turku, Turku, Finland*

Abstract

We examine the security features of LoRaWAN v1.1 and propose countermeasures for the determined security problems. LoRaWAN is among the emerging wireless communication technologies for the internet-of-things (IoT) that provide long-range connectivity for low-power IoT devices. As most IoT based applications operate without human intervention and deal with sensitive data, it is crucial to keep the security of LoRaWAN under scrutiny. The examined features in this work are key management, the newly introduced delegation of join procedure to network operators, backward compatibility, and replay protection for join procedure. The evaluation of key management exposes the fact that LoRaWAN v1.1 does not provide forward secrecy. The closer study of the join procedure delegation with backward compatibility reveals that they cannot securely coexist. The examination of join procedure demonstrates that when the assumption of trustworthy network server fails, not only integrity but also confidentiality of application data may be compromised. To overcome these issues, we proposed countermeasures that prevent the compromise of integrity and confidentiality of application data in the cases of join procedure delegation and malicious network server.

Keywords: LoRaWAN ; IoT ; security ; join procedure

1. Introduction

Internet of Things (IoT) is becoming ubiquitous through the deployment of IoT devices in virtually every corner of the economy, such as wearable, individual households, industries and vehicles. According to forecast, billions of devices will be interconnected using various connectivity technologies to sense and act on their environments in near future. A 2017 report by Ericsson [1] predicts that, by 2022, the number of wide-area connectivity (such as LTE-M, NB-IoT, Sigfox, LoRaWAN, and RPMA) based devices will reach 2.1 billion, and the number of short-range (such as Wi-Fi, Bluetooth, and ZigBee) IoT devices will reach 15.5 billion. The large number of interconnected devices, their limited resources, and the fact that they operate autonomously without human involvement, emphasize the importance of security in ways which clearly indicate that security can neither be ignored nor be an afterthought for IoT. IoT security is a multifaceted problem. The multitude of vulnerable IoT devices [2] is the embodiment of the failure of free-market in creating incentives for security. Furthermore, each newly introduced connectivity technology inevitably brings along its own set of vulnerabilities, and increases the overall security challenges of the IoT ecosystem. This work focuses on the security of one of the wide area wireless communication technologies, LoRaWAN [3].

* Corresponding author. Tel.: +358-4652-2-9553.
E-mail address: tcmdon@utu.fi

10.1016/j.procs.2018.07.202

Several Low Power Wide Area Network (LPWAN) technologies are emerging to provide connectivity for IoT applications, which require low-power long-distance wireless communication. These technologies often differ in their characteristics, such as range, power consumption, data rate, and latency. In terms of these characteristics and the trade-offs made between them, LPWAN complements traditional short-range wireless technologies [4]. LoRa$^{\text{TM}}$ is a wireless physical layer solution for long-range low-power low-data-rate applications developed by Semtech. LoRaWAN$^{\text{TM}}$ is a LPWAN specification optimized for wireless battery-powered end-devices based on LoRa. LoRaWAN specification version 1.0.2 [5] and 1.1 [6, 7] were released in July 2016 and October 2017, respectively. Previous works studied the security of LoRa-enabled systems. Vulnerabilities were discovered in LoRa physical layer [8, 9], specific implementations [10, 11], and in LoRaWAN protocol version 1.0 [12, 13, 14, 15]. Dönmez et al. [16] studies the security of version 1.1, but considers only the backward compatibility scenarios. In this paper, only LoRaWAN protocol vulnerabilities are of interest. At the time of writing, the authors were unable to find any work addressing the security of LoRaWAN version 1.1.

Though LoRaWAN version 1.1 has addressed most of the known security vulnerabilities in version 1.0.2 of the protocol, it is possible that not all existing vulnerabilities were discovered, or new vulnerabilities were introduced by version 1.1. The presented work examines the security of LoRaWAN version 1.1, and proposes countermeasures for some of the identified security issues. Examined features include key management, *join procedure delegation*, backward compatibility, and replay protection for the join procedure. Examination of key management exposes the fact that LoRaWAN does not provide forward secrecy. Consideration of the newly added *join procedure delegation* feature together with the backward compatibility of LoRaWAN reveals that they cannot securely coexist. Inspection of the join procedure demonstrates that the possibly misplaced trust in the network servers may lead to the compromise of not only the integrity but also the confidentiality of application data. The proposed countermeasures prevent the compromise of the integrity and confidentiality of application data in case of 1) *join procedure delegation* and 2) malicious network server.

The rest of the paper is organized as follows. Section 2 presents the key security features of LoRaWAN v1.1. Section 3 discusses known security issues in the previous versions of the protocol, in particular, their applicability to the current version and their possible extension to the new features added in version 1.1. Key and lifecycle management issues, and the lack of forward secrecy in LoRaWAN is discussed in Section 4. Section 5 discusses the security of LoRaWAN in backward compatibility scenarios, in particular, the effects of a fall-back on the version 1.1 *join procedure delegation* feature. Section 6 discusses the security implications of a missing replay protection in the join server. Finally, the conclusion is presented in Section 7.

2. Security overview: keys, counters, and join procedure

This section presents a brief overview of the security keys, their derivation, counters and nonces, and the join procedure of LoRaWAN version 1.1. LoRaWAN defines two activation methods, *Activation By Personalization* (ABP) and *Over-The-Air Activation* (OTAA). An ABP end-device (ED) is tied to a specific network during its manufacturing, and unlike an OTA activated ED, it does not go through a *join procedure*. In many cases, security has to be considered separately for ABP and OTAA EDs. OTAA is the more interesting case, and is also the recommended activation method for higher security applications [6].

2.1. Security keys and key derivation

Security of LoRaWAN is built on two AES-128 root keys, AppKey and NwkKey. Root keys are specific to an ED, and are stored by both the ED and its join server (JS). Root keys are used for deriving the two lifetime keys JSIntKey and JSEncKey, the three network session keys NetSKeys (SNwkSIntKey, FNwkSIntKey, NwkSEncKey) and the application session key AppSKey. Root keys are also used for protecting the integrity of Join-request messages, and protecting the confidentiality of Join-accept messages triggered by Join-request messages.

JSIntKey is employed for protecting the integrity of Rejoin-request messages of type 1 and Join-accept messages. JSEncKey is used for protecting the confidentiality of Join-accept messages triggered by Rejoin-request messages. SNwkSIntKey is used for protecting the integrity of data messages and Rejoin-request messages of type 0 and type 2. FNwkSIntKey is used in partial integrity checks of uplink data messages by the forwarding network servers (NS). NwkSEncKey is used for protecting the confidentiality of MAC commands. AppSKey is used for protecting the confidentiality of application data.

Session key derivation happens in both the ED and the JS. The two nonce values involved in the derivation of the four session keys are JoinNonce, and one of DevNonce, RJcount0, RJcount1, depending on the type of the message, which triggered the session key derivation.

ABP EDs are assigned session keys in the manufacturing stage, and the same session keys are used throughout the ED's lifetime.

2.2. Counters and nonces

The general strategy adopted by LoRaWAN to deal with the saturation of counters can be summarized as follows. In case of session counters, a security context switch is forced, resulting in the derivation of new session keys, and the resetting of session counters. In case of lifetime counters, recommendations are given for limiting the periodicity of increments, so that counters do not overflow within the planned ED lifetime. In addition to playing a role in the correct (or incorrect) use of the block cipher modes of operations, LoRaWAN's counters and nonces are also used for replay protection.

Frame counters: LoRaWAN uses three counters called *frame counters* for keeping track of uplink/downlink data messages. FCntUp counts uplink data messages, and is incremented with each uplink, except retransmissions. NFCntDown counts downlink data messages carrying only MAC commands. AFCntDown counts downlink data messages carrying application data (possibly along with MAC commands). FCntUp value is sent in FCnt field of uplink messages. Receiving network element synchronizes the FCntUp counter for the ED with the received value. Similarly, either NFCntDown or AFCntDown is sent in FCnt field of downlink messages, and the receiving ED synchronizes the corresponding local counter with the received value. In both cases, the synchronization occurs only if the message is authenticated, and the received counter value is incremented compared to the local counter value, which is the previously observed value. Frame counters are lifetime counters in case of ABP, and session counters in case of OTAA.

DevNonce and JoinNonce: DevNonce is a 16-bit counter managed by the ED. It is initially set to 0, and then incremented with every Join-request message sent. JoinNonce is a 24-bit counter. JS manages one JoinNonce counter per ED. JoinNonce is initially set to 0, and then incremented with every Join-accept message sent to the corresponding ED. DevNonce and JoinNonce are lifetime counters, and they must never repeat.

RJcount0 and RJcount1: RJcount0 and RJcount1 are 16-bit counters managed by the ED. RJcount0 is incremented with every type 0 or type 2 Rejoin-request message sent, and reset to 0 each time a Join-accept is successfully processed. RJcount0 must never repeat within a session. RJcount1 is a lifetime counter, and is never reset. It is initially set to 0, and then incremented with every type 1 Rejoin-request message sent.

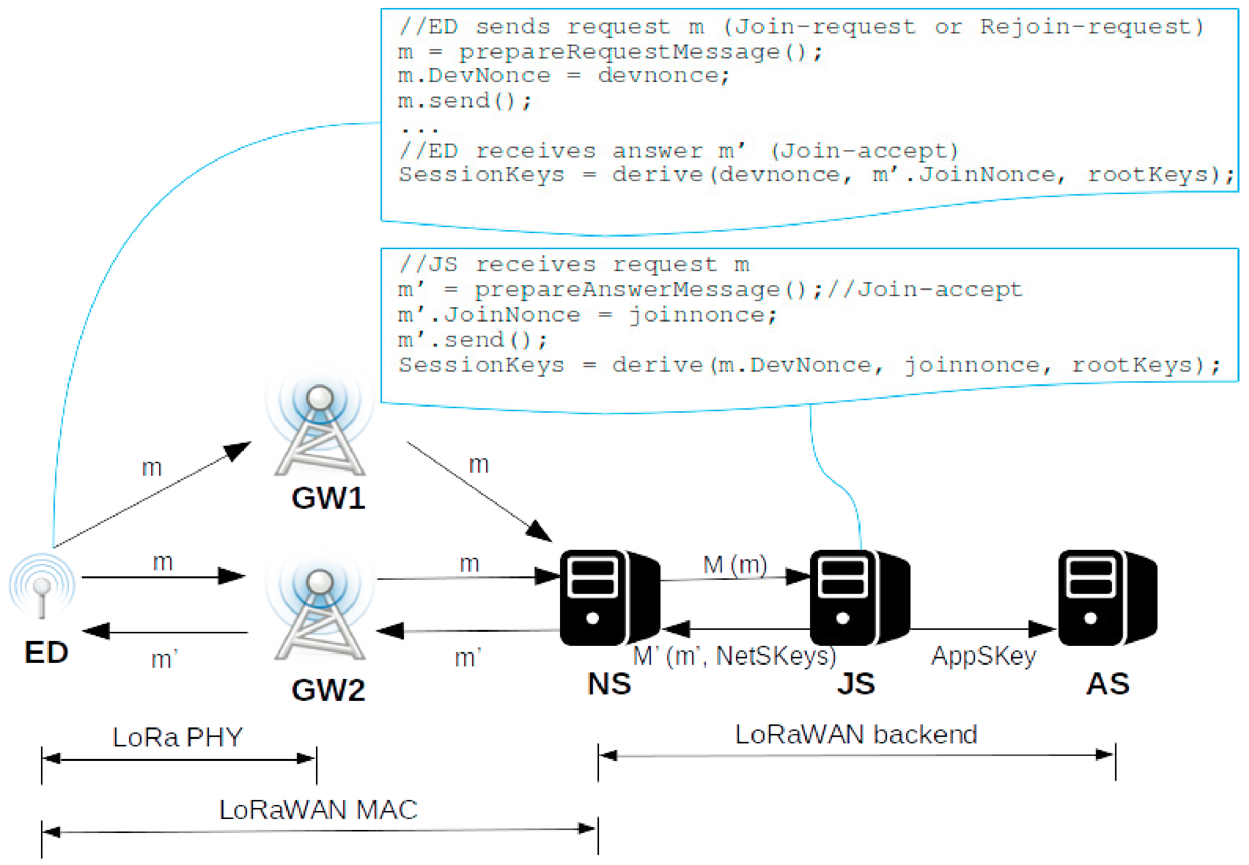
2.3. Join procedure

An OTAA ED must successfully complete a join procedure in order to be able to communicate with the NS. Join procedure involves the exchange of a Join-request (or Rejoin-request) and a *Join-accept* message between the ED and the JS. Processing of the request and the *Join-accept* message, as well as the derivation of the session keys happen at the ED and the JS. NS wraps, unwraps, and forwards request and answer messages between the ED and the JS. NS also decides on the MAC version to be used by the ED and the JS, and manages the security session context switch upon reception of the session keys. Figure 1 depicts LoRaWAN's join procedure for the *activation at home* scenario. It is also possible that the ED initiates the join procedure while roaming. The basic join procedure depicted in Figure 1 also applies to *roaming activation*, but the actual procedure is more involved in that case.

3. Applicability of v1.0 attacks to LoRaWAN v1.1

Dönmez et al. [16] presents a consolidated list of known protocol vulnerabilities in LoRaWAN version 1.0, and describes how each vulnerability was addressed in version 1.1. These v1.0 vulnerabilities lead to the following attacks: *Attack 1 - replay or eavesdrop*, *Attack 2 - ack spoofing*, *Attack 3 - replay or eavesdrop via fake session on ED*, *Attack 4 - replay or eavesdrop via fake session on NS*, *Attack 5 - DoS on ED via Join-accept replay*, *Attack 6 - DoS on ED via Join-request replay*, *Attack 7 - bit flipping* [12, 13, 16]. It is natural to start the study of v1.1 security by examining the vulnerabilities in v1.0 and the resulting attacks. This section examines the possible applicability and extensibility of these attacks to v1.1.

Attacks 1-6 are not applicable in LoRaWAN version 1.1, as the underlying vulnerabilities were addressed. Applicability of *Attack 7* in LoRaWAN version 1.0 depends on strong assumptions of malicious NS or NS-AS channel with missing or broken integrity protection. LoRaWAN specification assumes trust in network servers, in addition to secure channels between backend elements. While a malicious NS is an uncommon threat in the current paradigm, a paradigm shift may change this situation. For example, adoption of the micro operator concept [17] would possibly increase the likeliness of malicious network servers. Simply assuming trust in certain entities may limit the flexibility of the obtained IoT solutions in the face of future changes. Applicability of *Attack 7* in LoRaWAN version 1.1 is almost



```
//ED sends request m (Join-request or Rejoin-request)
m = prepareRequestMessage();
m.DevNonce = devnonce;
m.send();
...
//ED receives answer m' (Join-accept)
SessionKeys = derive(devnonce, m'.JoinNonce, rootKeys);
```

```
//JS receives request m
m' = prepareAnswerMessage();//Join-accept
m'.JoinNonce = joinnonce;
m'.send();
SessionKeys = derive(m.DevNonce, joinnonce, rootKeys);
```

Fig. 1. LoRaWAN's join procedure.

the same as in the older versions, except that the inclusion of handover roaming in v1.1 makes the situation worse. Handover roaming presents more possibilities for a man-in-the-middle attack, as the unprotected *FRMPayloads* are first transported from the serving NS to the home NS, and from there to the AS.

Due to the newly added *Rejoin-request* messages in v1.1, it is necessary to evaluate the extensibility of *Attacks 3-6* as they target LoRaWAN's *join procedure*. In order to examine the extensibility of these attacks, the possibility of modifying these attacks to work with the *Rejoin-request* messages and the associated nonces *RJcount0* and *RJcount1* has to be considered. If a *Rejoin-request* message is replayed, it will be discarded by the NS based on the last observed nonce value. NS only accepts nonce values which are incremented compared to the last observed value. Unlike in version 1.0, all LoRaWAN 1.1 nonces, including *RJcount0* and *RJcount1*, are counters. Prevention of reuse is practically possible in all cases, as only the last observed values have to be tracked. Furthermore, *Rejoin-request* messages are associated with the resulting *Join-accept* messages, which was not the case for *Join-request* messages in version 1.0. The association is achieved through the inclusion of the nonce value used for the request message,[1] in the MIC calculation for the *Join-accept* message. Finally, the inclusion of the request type *JoinReqType* in the MIC calculation for the *Join-accept* message prevents the replay of a captured *Join-accept* message, in response to a request of a different type. This inclusion is crucial because the counters *DevNonce*, *RJcount0*, and *RJcount1* will inevitably take on the same values. As the *Rejoin-request* messages and the associated nonces do not suffer from the vulnerabilities underlying *Attacks 3-6*, these attacks can not be modified in a straightforward manner to work with the new features introduced in version 1.1

[1] LoRaWAN specification for version 1.1 [6, p. 55] does not state that *DevNonce* should be replaced with *RJcount0* or *RJcount1* in *Join-accept* MIC calculation, when the type of the request is not *Join-request*. This is probably an accidental omission.

4. Root key compromise and forward secrecy

Physical access, key management, and life-cycle management are major challenges in IoT security. Deployment in non-monitored areas drastically increases the risk of physical tampering, while the end-device and network constraints make tasks such as re-keying and OTA firmware updates more difficult. The root keys stored in the end-devices are intrinsic to the overall security of LoRaWAN, and they must be protected against physical tampering. The specification leaves secure provisioning, storage, and usage of root keys out of scope, but points to SE (Secure Elements) and HSM (Hardware Security Modules) as possible enablers for a solution. LoRaWAN does not feature OTA firmware updates. This makes issuing fixes for the discovered vulnerabilities to the deployed end-devices impractical, and also makes it more likely that different versions of the protocol will coexist in a network, which in turn elevates the importance of addressing security issues related to backward compatibility.

LoRaWAN's *Join-request* and *Rejoin-request* messages which carry *DevNonce*, *RJcount0*, *RJcount1* values are not encrypted, and *JoinNonce* values can also be read once the *Join-accept* messages are decrypted using the compromised keys.[2] With the nonces and root keys at hand, an attacker can recover every used session key. Therefore, LoRaWAN (both the current and older versions) does not provide forward secrecy: session keys are not protected in case the root keys are compromised. The lack of forward secrecy elevates the importance of mitigating root key compromise. If the root keys of an ED are compromised at some point, an attacker who has sniffed and recorded past traffic can decrypt all the downlink/uplink application messages to and from that ED.

Kim et al. [14] proposed a key scheme to achieve forward secrecy. In their scheme, root keys are used only for an *initial join procedure* and then purged from the ED. For any subsequent activations, session keys of the previous session are used where normally the root keys would be used. In case of a device compromise, the only keys that can be recovered by an attacker are the session keys of the ongoing session. These keys cannot be used for deriving the session keys used in the previous sessions. Assuming the only change made to LoRaWAN v1.1 session key derivation is the replacement of the root keys with the corresponding session keys from the previous session, application session key for an ongoing session can be derived as follows:

$$AppSKey = aes128\_encrypt(LastAppSKey, 0x02|JoinNonce|JoinEUI|DevNonce^{3}|pad16)$$

Decrypting application traffic from the previous session requires the attacker to recover the application session key used in the previous session, *LastAppSKey*, from a single plaintext-ciphertext pair and a collection of ciphertexts encrypted with *LastAppSKey*. In this approach, it is still possible to recover from session losses on the NS. However, if the join server ever looses track of the last used session's key for a device, the result would be the same as the loss of root keys on the JS, and the device would not be able to perform joins or rejoins.

5. Backward compatibility and security

Backward compatibility is always a desired feature but it also has the potential to affect security in a negative way. As of March 2018, LoRaWAN version 1.0 devices and backends have been deployed in 100 countries by 76 network operators, and also by private providers and initiatives [18]. Lack of OTA firmware updates would make full migration to LoRaWAN v1.1 difficult, even if all the deployed end-devices were capable of running version 1.1. In case of LoRaWAN, potential security problems caused by backward compatibility are two-fold: a fall-back may enable the exploitation of vulnerabilities associated with the older version, or it may break a feature introduced in the new version. As exploitability of version 1.0 vulnerabilities in case of a fall-back are studied in our previous work in [16], the focus in this work is on assessing the effects of backward compatibility on the newly introduced *join procedure delegation* feature.

5.1. Join procedure delegation

In LoRaWAN v1.0, security of both network and application data depends on a single root key *AppKey*, whereas LoRaWAN v1.1 features two root keys *AppKey* and *NwkKey*. Apparently, the sole reason behind the introduction of the second root key is enabling the delegation of *join procedure* to network operators, without compromising the

[2] *Join-accept* messages are encrypted using *NwkKey* in case of version 1.0, rootkey *NwkKey* or lifetime key *JSEncKey*, in case of version 1.1
[3] *RJcount0* or *RJcount1* may replace *DevNonce* depending on the join request type.

confidentiality of the application data [6]. The delegation is achieved by surrendering *NwkKey* to the network operator. The network operator is not able to eavesdrop on application data because the application session key *AppSKey* is derived from *AppKey*, i.e. security of application data depends only on *AppKey*.

Introduction of a whole new root key suggests that *join procedure delegation* is an important use-case scenario to consider, however the LoRAWAN specification does not specify how a delegated *join procedure* works. We assume the following for *join-procedure delegation*:

- Part of version 1.1 JS functionality, namely *AppSKey* derivation, is carried out in a server in possession of *AppKey*, outside the control of the network operator. *AppSKey* derivation is carried out according to the MAC version decided by the NS, in accordance with the specification [6]. This server will be referred to as JS_A.
- The rest of version 1.1 JS functionality is carried out in a server under the control of the network operator. With *NwkKey* in hand, the network operator derives the network session keys, and constructs *Join-accept* messages, in accordance with the specification [6]. This server will be referred to as JS_N.
- The serving NS decides on the MAC version (*MACVersion*) in accordance with the specification [7]. The NS communicates the decision on MAC version to both JS_N and JS_A. The ED learns of the serving NS's decision via the *OptNeg* flag included in the *Join-accept* message constructed by JS_N.

The delegation of join procedure makes sense only when one assumes the existence of application sessions, which span several network sessions. If this is not the case, i.e. if the network session and the application session always start and end simultaneously, then a new AppSKey has to be derived each time new network session keys are derived. Clearly the derivation of AppSKey can not be delegated, unlike the other tasks involved in the join procedure which depend on the root key, NwkKey. So, the network operator can never handle a join procedure without the involvement of the party holding the root key, AppKey. However, nowhere in the specification application sessions that span multiple network sessions are explicitly mentioned. There is also no mention of any mechanisms that would enable such session scheme. For example, it is not possible for the ED to know when to derive a new AppSKey and when to keep using the old one, following the processing of a *Join-accept* message.

In another perspective, lack of forward secrecy has negative implications for join procedure delegation. If the surrendered root key *NwkKey* ever falls into the hands of an attacker, application data sent or received during the sessions involving a fall-back can be decrypted, even if the session took place before the compromise. When forward secrecy is achieved via the mechanism suggested by Kim et al. [14], delegation of *join procedure* to a network operator is still possible. Device owner surrenders the root key *NwkKey* to a network operator in case the *initial join procedure* has not yet taken place, and the last used network session keys, otherwise. However, it is up to the network operator to purge the root key (if received), and any session keys that are no longer needed in order to maintain the forward secrecy.

5.2. *Fall-back to version 1.0 breaks join procedure delegation*

The relevant backward compatibility scenario involves a v1.1 ED and a v1.0 NS[4]. We consider the case where a device owner surrenders the *NwkKey* of v1.1 ED to a network operator, assuming that this will not enable the operator to eavesdrop on application data. However, the fall-back mechanism for session key derivation described in the specification [6] invalidates the device owner's initial assumption about confidentiality. Furthermore, if the application implements own end-to-end integrity solution as suggested in the specification, end-to-end integrity is also broken, if the implemented integrity solution depends on *AppSKey*. In case of a fallback, all session keys, including *AppSKey*, are derived using the *NwkKey* of the version 1.1 ED as follows [6, p. 55]:

$$\begin{aligned}
&AppSKey = aes128\_encrypt(NwkKey, 0x02|JoinNonce|NetID|DevNonce|pad16)\\
&FNwkSIntKey = aes128\_encrypt(NwkKey, 0x01|JoinNonce|NetID|DevNonce|pad16)\\
&SNwkSIntKey = NwkSEncKey = FNwkSIntKey
\end{aligned}$$

[4] In LoRaWAN version 1.0 there is no separate element called a join server. The NS also serves as the JS.

Surrendering *NwkKey* to a network operator running a v1.0 backend does not make sense, and must be avoided by the device owner. However, it may also be the case that at the time of surrendering the *NwkKey* the network operator is running a v1.1 NS, but later reverts to v1.0.[5] Or the network operator still runs a v1.1 NS, but the NS, perhaps maliciously, decides on MAC version 1.0, even though the highest common version between the ED and itself is 1.1. Consequently, the ED will end up using its *NwkKey* for deriving the *AppSKey*.

Theoretically, handover-roaming may cause a fall-back when the backend that serves the roaming ED runs an older version. However, handover-roaming is itself a v1.1 feature, and there is no version to fall-back to for the current version 1.1, i.e. handover-roaming into a version 1.0 network is not possible. Handover-roaming involves the entrustment of only the network session keys. Unless the network operators take it a step further, and entrust the network root keys surrendered to them, to other operators they have roaming agreements with, handover roaming should not introduce further problems with regard to *join procedure delegation*.

5.3. Proposed countermeasure for eavesdropping

We propose two alternative countermeasures for preventing the network operator from eavesdropping on application data. The first countermeasure involves the alteration of *AppSKey* derivation mechanism, by introducing a special case for *join procedure delegation*. If the join procedure for the ED is delegated, JS_A derives *AppSKey* from *AppKey*, independent of the MAC version decided by the NS:

$$AppSKey = aes128\_encrypt(AppKey, 0x02|JoinNonce|JoinEUI|DevNonce|pad16)$$

The ED must derive *AppSKey* in the same way as JS_A does. One option is to have the ED preconfigured to always derive *AppSKey* from *AppKey*, independent of the received *OptNeg* flag value. The behaviours of JS_N and ED with regard to the derivation of the network session keys are still guided by the NS's decision on the MAC version. The second countermeasure introduces a mechanism which allows a device owner to configure the ED in order to restrict its operation to a minimum protocol version. Setting minimum protocol version to 1.1 would prevent eavesdropping of application data in the *join procedure delegation* case, at the expense of breaking backward compatibility of the v1.1 ED. The ability to set the minimum protocol version may also be desirable for other cases. For example, it would be possible to set minimum protocol version of the end-devices to 1.1 for security-sensitive applications, in order to prevent vulnerabilities associated with the older version from affecting them [16].

6. Missing replay protection in JS and countermeasure

The specification assigns the responsibility of keeping track of last observed *DevNonce* values to the NS [6, p. 53]. A malicious network server (or a man-in-the-middle attacker if channel integrity is compromised or non-existent) can easily cause a JS to saturate its 24-bit *JoinNonce* counter, by replaying the same *Join-request* message repeatedly. With each processing of the *Join-request* message, the JS increments the *JoinNonce* counter for the ED. If the JS implementation allows its *JoinNonce* counter to overflow and a reuse occurs, previously used session keys may be derived and used again, if *DevNonce* values also repeat.[6] Actually, if *JoinNonce* and *DevNonce* resets can be achieved simultaneously, all previously used session keys will eventually be reused. As the task of assigning device addresses lies with the (malicious) NS, inclusion of *DevAddr* within the encryption and MIC blocks does not pose a challenge to the attacker. Session key reuse breaks confidentiality and replay protection of data messages, among other things, as described in [12]. The specification leaves end-to-end integrity out of scope, partially based on the assumption that NSs are trustworthy. This case demonstrates that, end-to-end confidentiality of application data may also be compromised, if the NS is malicious.

The same replay protection vulnerability also applies to *Rejoin-request* messages of types 0 and 2, as the specification gives the responsibility of keeping track of *RJcount0_last* values to the NS [6, p. 58]. Consequently, a malicious NS can saturate the *JoinNonce* counter of the JS by replaying not only *Join-request* messages but also *Rejoin-request* messages sent for the purpose of rekeying, reassigning *DevAddr*, or handover roaming. As a result, the JS implementation enforcing a limit on the frequency of join requests will not be a solution.

[5] In this case, the NS would need added support for *join procedure delegation*.
[6] A repetition of *DevNonce* values can be caused for example by cutting power to a v1.0 ED.

We propose a countermeasure for protecting the application data from replay attack. In the proposed countermeasure, the JS keeps track of *DevNonce_last* and *Jcount0_last* values, and enforces the replay protection mechanisms described above, independently from the NS. In addition, a JS realizes the saturation of the *JoinNonce* counter and stops processing *Join-request* and *Rejoin-request* messages from the corresponding ED if the *JoinNonce* counter overflows.

7. Conclusion

In this work, security of LoRaWAN v1.1 was examined and countermeasures were proposed for the determined security issues. The security features included in the examination were key management, join procedure delegation, backward compatibility, and replay protection in join procedure. Though LoRaWAN v1.1 has addressed most of security vulnerabilities of v1.0, the evaluation in this work disclosed the lack of forward secrecy, flaw in delegation of join procedure in case of fall-back, and limitations in replay protection. Due to the lack of forward secrecy, the impact of root key compromise on application security is escalated. The investigation of the newly introduced delegation of join procedure revealed that integrity and confidentiality of application data may be compromised because of fall-back or malicious behavior of network server. The investigation of the replay protection mechanisms revealed that a malicious network server is able to replay OTA activation messages to the join server, and this may in turn causes the overflow of a counter nonce, and the compromise of the integrity and confidentiality of application data. The proposed countermeasures tackle the application data integrity and confidentiality violations in case of join procedure delegation and malicious network server.

Acknowledgements

This work is supported in part by Tekes under the project Wireless for Verticals (WIVE). WIVE is a part of 5G Test Network Finland (5GTNF).

References

[1] "Ericsson Mobility Report", June 2017 https://www.ericsson.com/assets/local/mobility-report/documents/2017/ericsson-mobility-report-june-2017.pdf

[2] Nicola Dragoni, Alberto Giaretta, and Manuel Mazzara. (2018) "The Internet of Hackable Things", in Paolo Ciancarini et al. (eds) *PROCEEDINGS OF 5TH INTERNATIONAL CONFERENCE IN SOFTWARE ENGINEERING FOR DEFENCE APPLICATIONS, Advances in Intelligent Systems and Computing*, Springer, Vol. 717, pp. 129–140. doi: 10.13140/RG.2.2.19643.72482

[3] "LoRa Alliance Technology", https://www.lora-alliance.org/technology

[4] U. Raza, P. Kulkarni and M. Sooriyabandara, "Low Power Wide Area Networks: An Overview," in IEEE Communications Surveys & Tutorials, vol. 19, no. 2, pp. 855–873, Secondquarter 2017. doi: 10.1109/COMST.2017.2652320

[5] LoRa Alliance Technical Committee. LoRaWAN^{TM}Specification, July 2016. LoRa Alliance, version 1.0.2.

[6] LoRa Alliance Technical Committee. LoRaWAN^{TM}Specification, Oct 2017. LoRa Alliance, version 1.1.

[7] LoRa Alliance Technical Committee. LoRaWAN^{TM}Backend Interfaces 1.0 Specification, Oct 2017. LoRa Alliance, version 1.0.

[8] E. Aras, G. S. Ramachandran, P. Lawrence and D. Hughes. (2017). "Exploring the Security Vulnerabilities of LoRa," in 3rd IEEE International Conference on Cybernetics (CYBCONF), Exeter, 2017, pp. 1-6. doi: 10.1109/CYBConf.2017.7985777

[9] E. Aras, N. Small, G. S. Ramachandran, S. Delbruel, W. Joosen, D. Hughes. (2017). "Selective Jamming of LoRaWAN using Commodity Hardware". doi: 10.1145/3144457.3144478.

[10] S. Tomasin, S. Zulian and L. Vangelista, "Security Analysis of LoRaWAN Join Procedure for Internet of Things Networks," 2017 IEEE Wireless Communications and Networking Conference Workshops (WCNCW), San Francisco, CA, 2017, pp. 1-6. doi: 10.1109/WCNCW.2017.7919091

[11] Notes on LoRaWAN security. Feb 1, 2017. URL: https://medium.com/@brocaar/notes-on-lorawan-security-7e741a8ee4fa (visited on 2017).

[12] Xueying Yang. "Lorawan: Vulnerability Analysis and Practical Exploitation". Delft University of Technology, 2017. URL: https://repository.tudelft.nl/islandora/object/uuid:87730790-6166-4424-9d82-8fe815733f1e?collection=education

[13] Gildas Avoine, Loïc Ferreira, "Rescuing LoRaWAN 1.0," unpublished. URL: https://fc18.ifca.ai/preproceedings/13.pdf

[14] Jaehyu Kim and JooSeok Song, "A Dual Key-Based Activation Scheme for Secure LoRaWAN," Wireless Communications and Mobile Computing, vol. 2017, Article ID 6590713, 12 pages, 2017. doi:10.1155/2017/6590713

[15] SeungJae Na, DongYeop Hwang, WoonSeob Shin and Ki-Hyung Kim, "Scenario and countermeasure for replay attack using join request messages in LoRaWAN," 2017 International Conference on Information Networking (ICOIN), Da Nang, 2017, pp. 718-720. doi: 10.1109/ICOIN.2017.7899580

[16] Tahsin C. M. Dönmez, and Ethiopia Nigussie, "Security of LoRaWAN v1.1 in Backward Compatibility Scenarios," accepted in The 15th International Conference on Mobile Systems and Pervasive Computing, 2018.

[17] Matinmikko, M., Latva-aho, M., Ahokangas, P. et al. Wireless Pers Commun (2017) 95: 69. https://doi.org/10.1007/s11277-017-4427-5

[18] "LoRaWAN NETWORKS", https://www.lora-alliance.org/