LoRaWAN1.1

Over The Air Activation - OTAA

Stefano Milani <stefano.milani96@gmail.com>
Ioannis Chatzigiannakis <ichatz@diag.uniroma1.it>

Abstract

[[TODO]]

Ⅰ - Introduction

[[Draft of introduction: to modify during the production of the report]]
[[To add: what we expect from a key agreement protocol. Refers to the follow presentation:https://github.com/StefanoMilani/RIOT-OS-examples/blob/doc/GroupKeyManagement.pdf]]
To ensure a secure communication between the end-device and the Join/Application Server, each device joining a LoRaWAN network has to be activated. There exists two types of activation:

- **Activation By Personalization (ABP):** This activation method is simpler, but less secure. It makes use of a set of predetermined keys, used to determine the session and integrity keys. These predetermined keys are hard-coded into the device, so in order to refresh the keys you need to manually change these keys in the code.
- **Over The Air Activation (OTAA):** The focus of this report is on this activation method. It permits the session and integrity keys to be agreed upon the end-device and the Join/Application server through a handshake. Also OTAA makes use of some hard-coded information into the device, that are used to derive the needed keys.

The rest of the report is organized as follows. A detailed presentation of the LoRaWAN1.1 OTAA method is presented in Section Ⅱ; the weak points and possible vulnerabilities of the activation method, referring to different attack models are listed in Section Ⅲ. A possible approach to mitigate some of these vulnerabilities is presented in Section Ⅳ; in Section Ⅴ are listed the result in terms of power and computational expense of the proposed model.

Ⅱ - LoRAWAN1.1 OTAA

[[Refers to OTAA presentation, basically extend the info already written there https://github.com/StefanoMilani/RIOT-OS-examples/blob/doc/OTAA.pdf]]

Ⅲ - Possible vulnerabilities

[[List of observations, obviously not final]]

First attack model: Passive Man In The Middle Attack. The attacker can intercept all messages and read the content if unencrypted. So basically he can read the content of everi Join-Request and Rejoin-Request message.

The useful information for an attacker are the one useful to create the session and the session-integrity keys:

- **NwkKey**
- **JoinNonce**
- **JoinEUI**
- **DevNonce**

In the case of a Join-Request message, he knows the *DevEUI* and the *JoinEUI*, these information are already public, are needed to identify a specific device and a specific Join Server in the network. Also he knows *DevNonce*.

In case of a Rejoin-Request message of type 0 or type 2, an attacker can know the *NetID,* again the *DevEUI*, and the *RJcount0*. For a type 1 Rejoin-Request he knows the *JoinEUI*, the *DevEUI*, and the *RJcount1*. The two counters (*RJcount0* and *RJcount1*) are used to replace the *DevNonce* after a Rejoin-Request.

So performing a passive Man In The Middle Attack an attacker can know at each time of the lifetime cycle of a LoRa device the *JoinEUI* and the *DevNonce*.

[[In the LoRaWAN1.1 specification the JoinNonce is seen as counter, to check in the implementation if it is really a simple counter starting from 0]]
As we seen in Section Ⅱ, the *JoinNonce* is a counter specific for each device that the Join Server increments at each Join-Request or Rejoin-Request. An attacker can easily guess the *JoinNonce* by counting the number of the Join-Accept message. ***[[Understand if it is possible to know if a message is a Join-Accept message even if encrypted, maybe with the routing information?]]***

In conclusion to this attack model we can say that an attacker can retrieve a lot of info simply by passively intercept the message between an end-device and a Join Server, in particular the security of the OTAA protocol is based only on the correct storage of the **NwkKey** and the **AppKey**.

The **NwkKey** and the **AppKey** are aes128 root keys that we need to store into the device during fabrication, and during the entire lifecycle of the end-device it never changes. The LoRaWAN devices are supposed to work for 10 years, and these keys are never refreshed in these ten years. That could be a vulnerability of the OTAA protocol, since these are the only information that an attacker misses to compute the session and the integrity keys.

Moreover, in case of leak of these keys, the only way to recover is to shut down the device and to eliminate it in the Join and Application Server, because it is needed to generate a new pair of **NwkKey** and **AppKey**, and then insert them manually into the end-device. This process can be expensive and hard in particular in case of devices placed in hard reachable places.

Ⅳ - Using ECC to refresh the root keys