Technical Recommendations for Preventing State Synchronization Issues around LoRaWAN™ 1.0.x Join Procedure

NOTICE OF USE AND DISCLOSURE

Technical Recommendations for Preventing State Synchronization Issues around LoRaWAN™ 1.0.x Join Procedure

Authored by the LoRa Alliance Technical Committee

Chairs:
N.SORNIN (Semtech), A.YEGIN (Actility)

Editor:
A.YEGIN (Actility)

Contributors:
A.YEGIN (Actility), J.DELCLEF (ST Microelectronics), M.LE GOURRIEREC (Sagemcom)

Version: 1.0.0
Date: August 2018
Status: Final version

Contents

1 Introduction

This document describes potential End-device and network synchronization issues related to the LoRaWAN 1.0.x [LW10, LW102] Join (Activation) Procedure, and suggests how they can be overcome for a given deployment by implementing additional behavior on the end-points. Some of these issues could be leveraged by an adversary to prevent a target End-device from joining the network, unless they are addressed by the recommended solutions in this document.

The additional behavior recommended in this document is compatible with the LoRaWAN 1.0.x specification, therefore no change is required at the MAC layer. However, some of the remedies include additional behavior both at the End-device and the Join Server, which are expected to be configured synchronously. No in-band mechanism is defined to ensure such synchronization, except for defining the configuration attributes for the proposed behaviors. Careful configuration of both sides are the End-device and Join Server administrator's responsibility.

The identified issues do not exist for LoRaWAN 1.1 specification [LW11] as the proceeding remedies are already integral part of that specification.

2 Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

3 Issues

Following issues are identified as stemming from LoRaWAN 1.0.x specification:

Issue #1: Join-accept replays

The End-device sends a variance in the Join-request, namely DevNonce, but this value is not used in generation of the Join-accept frame. As such, the End-device has no way of knowing whether the received Join-accept is generated in response to the last Join-request it sent or not.

Unless the End-device keeps track of all Join-accept frames received in the past, it may be subjected to a replay attack. It is not always practical for an End-device to keep track of all Join-accept frames received in the past. Under this circumstance, an adversary can eavesdrop over the air for a while and collect Join-accept frames sent to an End-device. When the adversary replays one of those Join-accept frames to the End-device in response to a Join-request generated by the End-device, the End-device may accept this replay (e.g., due to actual Join-accept frame being absent or having lower signal strength). Such an End-device would fall out of synch with the network due to using session keys generated with an incorrect AppNonce.

Issue #2: Join-request replay protection challenges

In order to detect Join-request replays, the network is expected to keep track of history of (pseudo-random) DevNonce values used by a given End-device.

Unless the network keeps track of all DevNonce values ever used by an End-device, it is subject to Join-request replay attacks by an adversary that can track more Join-request frames than the network.

Keeping track of all used DevNonces is not practical. Furthermore, the more DevNonce values the network tracks, the higher the chance that a new Join-request will fail due to increased chances of colliding with an already used value when the DevNonce is generated randomly.

The LoRaWAN 1.0.x spec does not allow the Network Server to roll over to the new security context until the first uplink frame using the new context is received from the End-device following transmission of a Join-accept. But there is a possibility that this new frame is delayed (because the End-device has not received the Join-accept frame, or the uplink frame is lost, or the Join-request was really a replay coming from an adversary). The network needs to process new Join-requests that arrive even before a first uplink frame is sent by the End-device following the last Join-accept. This would either require the network to keep track of multiple but still a finite set of pending context, or force the network to invalidate previous pending context (the one still awaiting first uplink frame). That invalidation can be source of problem when the invalidated context was generated by an authentic Join-request and the later Join-request was a replay. This can keep the victim End-device out of the network.

Issue #3: Need for rapid security context switch for Class C

An End-device in Class C mode may have a downlink frame queued in the network at any time. If the network has sent a Join-accept to the End-device but has not received the associated first uplink frame which confirms the receipt of the Join-accept, the network would not know which security context to use when sending a Class C downlink frame to the End-device since it does not know if Join-accept was received by the End-device.

Note that these issues do not exist in LoRaWAN 1.1 deployments as the remedies described in the next section are already integral part of that newer specification.

4 Remedies

Following remedies are recommended for addressing the issues identified in the previous section.

Remedy #1: Preventing Join-accept replays

Remedy #1, Part 1 – Join Server Behavior:

> The Join Server SHALL provide the AppNonce as a device specific counter value (that never repeats itself) and that is incremented with every Join-accept message.

Remedy #1, Part 2 – End-device Behavior:

> The device is expected to be aware of this behavior of its Join Server and SHALL keep track of the AppNonce value used in the last successfully processed Join-Accept (corresponding to the last successful key derivation). The device SHALL accept the Join-accept only if the MIC field is correct and the AppNonce is strictly greater than the recorded one. In that case the new AppNonce value replaces the previously stored one.
>
> If the device is susceptible of being power cycled the AppNonce SHALL be persistent (stored in a non-volatile memory).

For a complete implementation of this remedy, it is expected that Part 1 is implemented by the Join Server before Part 2 is implemented by the End-device. It is RECOMMENDED that the Join Servers implement Part 1 as soon as possible independent of the availability of Part 2 implementations on the End-devices.

Recommended name for Join Server attribute to indicate implementation of this remedy is AppNonceCounter. When set to True, that means the Join Server is using counter-based AppNonce value.

Remedy #2: Simple and robust Join-request replay protection

Remedy #2, Part 1 – End-device Behavior:

> The End-device SHALL use a DevNonce value that is based on a counter starting at 0 when the End-device is initially powered up and incremented with every Join-request. A DevNonce value SHALL NOT be reused for a given AppEUI value. If the end-device can be power-cycled then DevNonce SHALL be persistent (stored in a non-volatile memory). Resetting DevNonce without changing AppEUI will cause the network to discard the Join-requests of the device.
>
> If there is a possibility that the End-device may need to generate more than 65,536 Join-requests in its lifetime, then the End-device SHALL be configured with multiple AppEUIs, and the End-device SHALL switch to another AppEUI upon consuming the whole DevNonce space of a given AppEUI.

Remedy #2, Part 2 – Join Server Behavior

> The Join Server is expected to be aware of this behavior of its End-devices and keep track of the last DevNonce value used by the End-device for the given AppEUI, and ignore Join-requests if DevNonce is not incremented.
>
> The JS SHALL keep incrementing the AppNonce monotonically according to Remedy #1 Part 1 even after switching the AppEUIs. This is because the NwkSKey/AppSKey derivation formulas take AppKey, NetID, DevNonce, and AppNonce into account, where AppKey and NetID have fixed values across Join procedures, and DevNonce can repeat itself in between Joins with different AppEUIs. Ensuring uniqueness on AppNonce ensures distinct NwkSKey/AppSKeys across Join procedures. The JS SHALL NOT reuse the same AppNonce value with the End-device, an assumption that we can take given the size of that value (24bits yields more than 16 million Join procedures).

For a complete implementation of this remedy, it is expected that Part 1 is implemented by the End-device before Part 2 is implemented by the Join Server. It is RECOMMENDED that the End-devices implement Part 1 as soon as possible independent of the availability of Part 2 implementations on the Join Servers.

Recommended name for End-device attribute to indicate implementation of this remedy is DevNonceCounter. When set to True, that means the End-device is using counter-based DevNonce value.

Remedy #3: Rapid key confirmation

The End-device that expects to receive Class C downlink frames SHALL send a confirmed uplink frame or a frame that requires an acknowledgement as soon as possible after receiving a valid Join-accept frame. The End-device SHALL continue to send such frames until it receives the first downlink from the network (while respecting duty cycles, if applicable, and retransmission timers). If the End-device does not receive an acknowledgement within the first ADR_ACK_LIMIT uplinks, then it SHALL revert to the Join state (i.e., sending another Join-request immediately or at a later point in time).

Even when the window of uncertainty is narrowed down for an End-device implementing this remedy, there is still a possibility that the Network Server has a downlink frame to send but the first uplink frame is not received. Whether the network opportunistically transmits the downlink frame or waits for the first uplink is an implementation/deployment decision.

If the network decides to transmit the downlink, it SHALL use the active/confirmed security context if Remedy #2 is not used, because there is a chance that the Join-request might be a replay sent by an adversary. The network SHALL use the pending context for an End-device using Remedy #2 when it decides to transmit the downlink frame before it receives the first uplink frame, as there is no chance of a replay.

Glossary

ABP	Activation by Personalization
ADR	Adaptive Data Rate
API	Application Programming Interface
AS	Application Server
DNS	Domain Name Server
ED	End-device
fNS	Forwarding Network Server
GW	LoRa Gateway
HTTP	HyperText Transfer Protocol
hNS	Home Network Server
IP	Internet Protocol
JS	Join Server
JSON	JavaScript Object Notation
KEK	Key Encryption Key
LoRa™	Long Range modulation technique
LoRaWAN™	Long Range network protocol
MAC	Medium Access Control
MIC	Message Integrity Code
NAPTR	Naming Authority Pointer
NS	Network Server
OTA	Over-the-Air
RF	Radio Frequency
RSSI	Received Signal Strength Indicator
SF	Spreading Factor
SIP	Session Initiation Protocol
SNR	Signal-to-Noise Ratio
sNS	Serving Network Server

Bibliography

References

[LW10] LoRaWAN Specification, Version 1.0, LoRa Alliance, January 2015.
[LW102] LoRaWAN Specification, Version 1.0.2, LoRa Alliance, July 2016.
[LW11] LoRaWAN Specification, Version 1.1, LoRa Alliance, October 2017.

NOTICE OF USE AND DISCLOSURE