A Simple and Efficient Replay Attack Prevention Scheme for LoRaWAN

Jaehyu Kim
Department of Computer Science,
Yonsei University
50 Yonsei-ro Seodaemun-Gu, Seoul,
03722, Republic of Korea
+82 (02) 21237269
jaehyu_kim@yonsei.ac.kr

JooSeok Song
Department of Computer Science,
Yonsei University
50 Yonsei-ro Seodaemun-Gu, Seoul,
03722, Republic of Korea
+82 (02) 21237269
jssong2526@yonsei.ac.kr

ABSTRACT
In recent years, LPWAN technology, designed to realize low-power and long distance communication, has attracted much attention. Among several LPWAN technologies, Long Range (LoRa) is one of the most competitive physical layer protocol. Long Range Wide Area Network (LoRaWAN) is an upper layer protocol used with LoRa, and it provides several security functions including random number-based replay attack prevention system. According to recent studies, current replay attack prevention of LoRaWAN can mislead benign messages into replay attacks. To resolve this problem, several new replay attack prevention schemes have been proposed. However, existing schemes have limitations such as not being compatible with the existing packet structure or not considering an exceptional situation such as device reset. Therefore, in this paper, we propose a new LoRaWAN replay attack prevention scheme that resolves these problems. Our scheme follows the existing packet structure and is designed to cope with exceptional situations such as device reset. As a result of calculations, in our scheme, the probability that a normal message is mistaken for a replay attack is 60-89% lower than the current LoRaWAN. Real-world experiments also support these results.

CCS Concepts
• **Security and privacy → Network security → Security protocols**

Keywords
LoRaWAN; LoRaWAN Security; Replay Attack; LPWAN; Internet of Things

1. INTRODUCTION
In recent years, a new type of wireless communication technology, called Low Power Wide Area Network(LPWAN), has emerged. LPWAN is designed to enable long-distance wireless communication with low power consumption. With this technology, battery-limited devices such as small sensors can communicate over long distances. Since it has not been properly supported by conventional network technologies, LPWAN technology is attracting much attention. In this context, various LPWAN technologies such as SigFox, Weightless, LoRa and Ingenu have been proposed [1]. Among them, LoRa is one of the most promising technologies with strengths in terms of deployment cost and power consumption [2]. LoRa supports long distance wireless communication up to 15 km with low-power consumption by using chirp spreading spectrum modulation. LoRaWAN is the upper layer protocol used with LoRa and it defines an overall system architecture and specific behavior of each entity.

As shown in Fig. 1, LoRaWAN environment consists of end nodes, gateways and server-side. The operation of each entity is defined in the LoRaWAN Specification [3]. The LoRaWAN end node transmits data to the gateways via the LoRa radio link. The LoRaWAN gateway transmits the packet received through the LoRa radio link to the network server via the conventional IP connection. The network server manages the entire LoRaWAN network. Upon receipt of the packet, it verifies the packet through the security process and sends an acknowledgement packet if necessary.

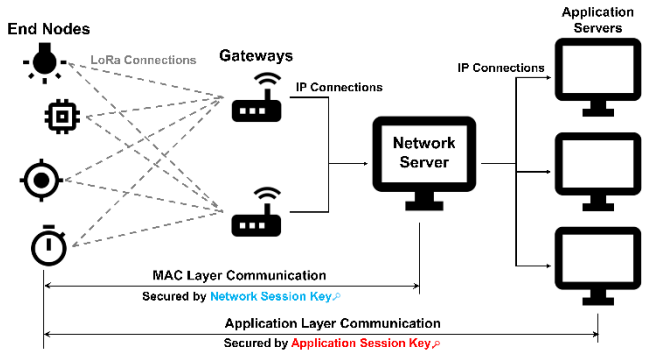

Figure 1. LoRaWAN Network Environment

LoRaWAN also provides basic security functions such as data encryption, message integrity checking, and end node authentication. The security of LoRaWAN communication is based on two session keys. One is the Network Session Key (NwkSKey) and the other is the Application Session Key (AppSKey). NwkSKey is used to secure the MAC layer communication between the network server and the end node. AppSKey is used to end-to-end encryption between the application server and the end node. The end node goes through the activation process to have the session keys. LoRaWAN

ICCNS 2017, November 24–26, 2017, Tokyo, Japan.

DOI: https://doi.org/10.1145/3163058.3163064

provides two methods for the end node activation. First one is the Activation by Personalization(ABP) method, which directly puts the session keys on the end node. The other is the Over The Air Activation(OTAA) method, which generates the session key through the join procedure between the end node and the network server.

In LoRaWAN join procedure, join request and join accept packets are used. Packet formats are depicted in Fig. 2. The end node initiates the join procedure by sending a join request packet to the network server. On the receipt of the join request packet, the network server generates the session keys. After that, the network server sends the join accept packet and it includes the parameters used in session key generation. After receiving the join accept packet, the end node can generate the session keys by using these parameters. As a result, after the join procedure is completed, the end node and the network server share the session keys. Since the join procedure is performed remotely via wireless communications, they are exposed to security attacks. Therefore, LoRaWAN protects the join procedure by applying various security mechanisms such as data encryption, message integrity checking, and replay attack prevention.

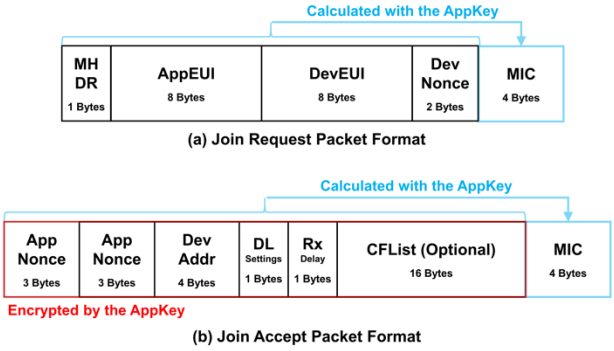

Figure 2. Join Procedure Packet Formats

In this paper, we focus on LoRaWAN's replay attack prevention system. As shown in [6], an attacker can capture a join request packet between the end node and the gateway. Then the attacker can perform the replay attack by retransmitting the captured join request packet to the gateway. If the replay attack is successful, the target node falls into the Denial of Service(DoS) state. In the current LoRaWAN, the network server uses DevNonce, a 16-bit random number included in the join request, to determine whether the packet was transmitted in the past. To do this, the network server should store and properly manage the DevNonces used in the previous join procedures. However, since the LoRaWAN Specification does not specify how to manage the DevNonce, some network servers may have problems in their DevNonce management system [4]. Furthermore, because of the short length of 16 bits, the probability that the benign node accidentally regenerates the previous used DevNonce cannot be ignored [4, 5]. To reduce this probability, a method to increase the length of the DevNonce field to 24 bits is proposed in [5]. However, this is not compatible with the current LoRaWAN because it requires the packet structure to be changed and additional overhead is also caused by the increased packet length. Another token-based approach, proposed in the [6], cannot cope with the case where the NwkSKey is lost on the end node.

Thus, in this paper, we try to resolve the limitations of the previous works and propose a new LoRaWAN replay attack prevention scheme. In the proposed scheme, we separate the current join request and newly define the initial join request and the non-initial join request. The non-initial join request is used in general cases and the replay attack of the non-initial join request packet is prevented based on the NwkSKey. Since the NwkSKey changes each time the join procedure is completed, the network server can use it to detect replay attacks. In case of the NwkSkey loss, end node uses the initial join request. Network server uses the current LoRaWAN DevNonce system to prevent the replay attack of the initial join request packet. As a result, our scheme works even in the case where the NwkSKey is lost on the end node due to the various reasons such as device reset. Furthermore, since our scheme follows the current join request packet format, it is compatible with the current LoRaWAN and has no additional transmission overhead. We evaluate the proposed scheme through the mathematical approach and real-world experiments. Through the mathematical approach, we confirm that the proposed scheme is more secure than the existing approaches. Compared to the current DevNonce system, proposed scheme has 97% lower probability that the benign node accidentally regenerates the previously used DevNonce in a year. Real-world experiments also support this result.

The rest of this paper is organized as follows. The rest of this paper is organized as follows. In section 2, we provide related works about the LoRaWAN replay attack. Section 3 is problem statement. In section 4, detailed description of the proposed scheme is provided. Section 5 is an evaluation of our scheme. Finally, in section 6, we provide a conclusion of this research.

2. Related Works

In [4, 5], the authors firstly showed that current DevNonce-based replay attack prevention have several problems. The first problem mentioned by the authors is about the case where the network server stores a certain number of DevNonces that have been transmitted in the past. For example, suppose a network server stores up to 10 recently transmitted DevNonces for replay attack prevention. In this case, when the 11th join procedure is completed, the network server drops the DevNonce of the first join request from the list to save the DevNonce of the 11th join request. As a result, the network server stores the DevNonce from the second to the 11th. At this point, if the attacker replays the first join request that was captured in the past, the network server compares the stored DevNonces with the DevNonce of the first join request. However, the network server removed the first DevNonce from the list, so it cannot detect the replay attack. To prevent such attacks, the network server should store all DevNonces used in the previous join procedure. However, if all DevNonces are stored on the network server, we must consider the situation where normal join requests are mistaken for replay attacks. This is because as the number of DevNonce stored in the network server increases, the probability that a generated DevNonce from the benign node is already stored in the network server is also increased. According to [4, 5], if an end node performs a join procedure once in a day, the probability of generating the previously used DevNonce in a year is about 98%. If the network server uses a policy that simply drops packets when the same DevNonce is sent, it may not be a big problem. However, if the network server takes a policy that excludes the node from the network when the same DevNonce is sent, a benign node can be accidentally excluded from the network. This is the second problem mentioned by the authors in the [4, 5]. It occurs because the current 16-bit length DevNonce field is not long enough to be

mathematically safe. To resolve this problem, the authors proposed to increase the DevNonce field to 24 bits or 32 bits in [4]. However, this method is not compatible with the current LoRaWAN because it changes the structure of the join request packet.

As another alternative to the current DevNonce method, Na et al. proposed a token-based solution in [6]. The token is created using a portion of the NwkSkey and is XOR-ed with the DevNonce and MIC fields of the join request packet. Since the NwkSKey changes each time the join procedure is completed, the network server can detect a replay attack using the token. This token-based approach is simple and effective, but there is no consideration for situations where the NwkSKey is lost on the end node. In LoRaWAN environment, NwkSKey data can be lost on the end node due to various reasons such as device reset or device malfunction.

3. Problem Statement

In this section, based on the limitations of current DevNonce and other previous works, we summarize the main problems that we are trying to resolve in this paper.

First Problem The first problem is that the join request sent by the benign node can be mistaken as the replay attack under the current DevNonce system. As mentioned in [4, 5], if the network server stores only some of the DevNonces used in the past, it can be vulnerable to replay attacks. Therefore, the network server needs to store all the DevNonces used in the previous join procedures. However, under the current 16-bit long DevNonce system, we cannot ignore the probability that the benign node will accidentally create the same DevNonce as already stored on the network server. The new replay attack prevention scheme should resolve this risk and not determine a normal join request as a replay attack.

Second Problem The second problem is about the compatibility with the current LoRaWAN. The 24-bit DevNonce scheme proposed in [4] to resolve the first problem is not compatible with the current LoRaWAN because it modifies the packet structure of the join request. Therefore, a new approach compatible with the current LoRaWAN is needed.

Third Problem The token-based approach proposed in [6] effectively prevents replay attack by using the NwkSKey, and it does not change the packet structure of the join request. However, it does not consider the situation where the NwkSKey data is lost in the end node. Therefore, if an event such as a device reset which can cause the NwkSKey loss occurs, the token-based scheme no longer works. This is the third problem that we want to resolve. The newly proposed replay attack prevention scheme should work well even if the NwkSKey data is lost.

4. Proposed Scheme

The main difference between previous works and our scheme is that there are two kinds of join requests. We newly define the initial join request and the non-initial join request. In our scheme, the end node determines which join request to use based on the presence of the NwkSKey. The network server applies different replay attack prevention methods depending on the type of join request.

4.1 Initial Join Request

The initial join request packet of our scheme is the same as the join request packet of the current LoRaWAN. It is designed for the case where there is no available NwkSKey on the end node. For example, when the device is turned on for the first time, or when the NwkSKey data is lost due to the device reset.

4.2 Non-Initial Join Request

In our scheme, non-initial join request packets are used in normal situations. The only difference between the non-initial join request packet and the initial join request packet is the key used to calculate the Message Integrity Code (MIC). As shown in Fig. 2, in the current LoRaWAN, Application Key (AppKey) is used to calculate the MIC of the join request. In our scheme, the NwkSKey is used to calculated the MIC. Since the NwkSKey is essentially the same 16-byte long key as the AppKey, it can also be used to calculated the MIC. The structure of the non-initial join request packet is the same as the initial join request packet.

4.3 Replay Attack Prevention Process

Non-Initial Join Request For non-initial join packets, the NwkSKey is used to prevent replay attacks. In our scheme, the MIC of the non-initial join packet is calculated using the NwkSKey. The valid NwkSKey changes each time the join procedure completes. Therefore, even if the attacker attempts to replay attacks with previously transmitted non-initial join request packets, the NwkSKey used for the packet is invalid and the attack cannot succeed.

Initial Join Request In the case of initial join request packets, the AppKey is used for MIC calculations, and the AppKey does not change over the lifetime of the end node. Therefore, a replay attack prevention method other than non-initial join request should be used. Our scheme uses the DevNonce for replay attack prevention of the initial join request packet, which is also used in current LoRaWAN. The problem of the current DevNonce is that the benign node can accidentally create the same DevNonce that is already stored on the network server. This probability increases as the number of DevNonces stored on the network server increases. Despite the existence of this risk, we can apply the current DevNonce method because the initial join request rarely occurs. Initial join requests are devised for the exceptional situations where the NwkSKey is lost, and it is reasonable to assume that these situations are very rare for an ordinary LoRaWAN end node. Thus, with regard to the probability that DevNonce will accidentally duplicated, initial join request of our scheme is mathematically more secure than the current LoRaWAN join request. We prove this through the evaluation provided in the next section.

Fig. 3 depicts the whole proposed replay attack prevention process.

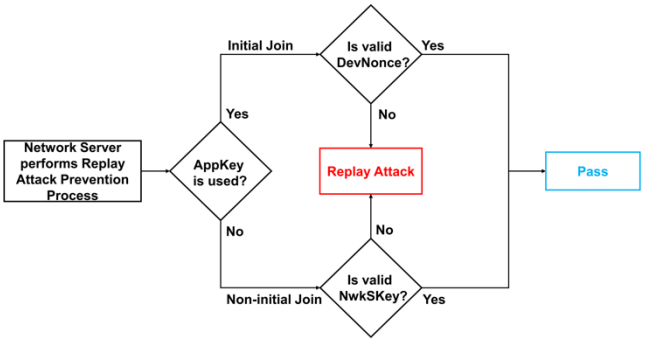

Figure 3. Proposed Replay Attack Prevention Process

5. Evaluation

In our scheme, an end node uses a 16-byte long NwkSKey to compute the MIC value of the non-initial join request. It is clear that the replay attack can be prevented because the NwkSKey changes every time the join procedure completes. However, our initial join request uses the current DevNonce method which can cause the unintentional duplication. Thus, as to the probability of the accidental duplication, the safety of our initial join request needs to be proved. In this section, we provide the mathematical calculation results and real-world experiments results.

5.1 Calculation

In [4, 5] the authors provide various mathematical formulas related to the current LoRaWAN DevNonce. We use them to compare the proposed scheme, current LoRaWAN DevNonce, and 24-bit DevNonce to each other.

According to [4, 5], the probability $\mathcal{E}(K)$ that the previous generated DevNonce will be generated at *K*th join request on an end node is calculated as follows:

$$\mathcal{E}(K) = \mathcal{D}(K-1) \cdot \frac{K-1}{N}$$

In this formula, $\mathcal{D}(K)$ means the probability that *K* different DevNonces will be generated consecutively. *N* is the total number of available DevNonces. In current LoRaWAN, it is set to 2^{16}=65536 because DevNonce field is 16-bit long.

As a result, if the network server stores all DevNonces, the probability of DevNonce duplication occurring in *K* join attempts is calculated as follows.

$$PR_{dup}(K) = cdf(\mathcal{E}(K))$$

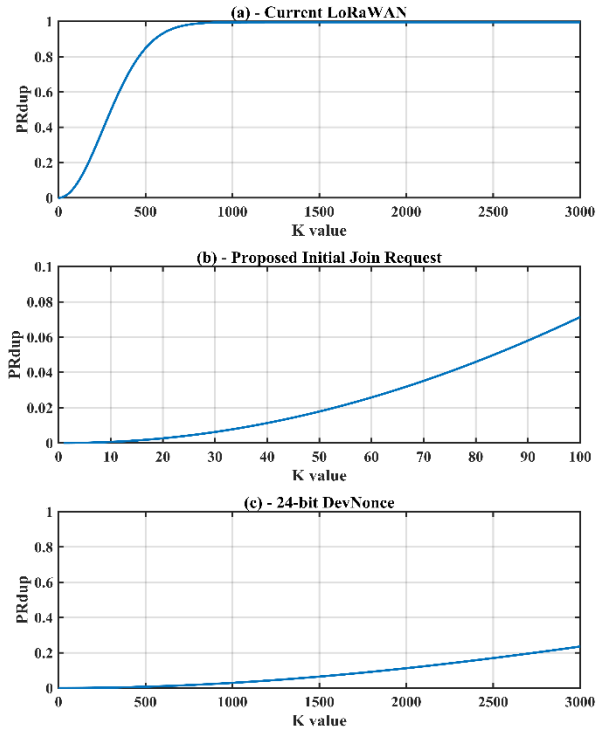

Figure 4. Probability that DevNonce duplication occurs within K attempts – Calculation Result

The calculation result of $PR_{dup}(K)$ according to the *K* value is shown in Fig. 4. As with the previous works [4, 5], we assume that the join procedure is performed once a day under the current LoRaWAN and 24-bit DevNonce. In case of the proposed initial join request, we assume that it is performed once a month considering that it is used only in exceptional cases. Since the lifetime of a LoRaWAN end node is generally expected to be at most 5-10 years [7], under the current LoRaWAN and 24-bit DevNonce, the end node performs 1500 - 3000 join procedures for a life time. In the same way, the initial join request of our scheme is performed about 50-100 times. Therefore, in case of current LoRaWAN and 24-bit DevNonce, the probability for 0 to 3,000 K values are shown in Fig. 4. And in case of the proposed initial join, K values from 0 to 100 are indicated.

Table 1 shows the probability of DevNonce duplication occurring within X years using calculation results in Fig. 4. In the first year, the probability that a normal node generates the same DevNonce used in the past is about 63% for current LoRaWAN, 0.39% for 24-bit DevNonce, and about 0.08% for proposed initial join. Proposed initial joins recorded the lowest probability, which is about 60% lower than current LoRaWAN and about 0.3% lower than 24-bit DevNonce. If the period is extended to 10 years, the proposed initial join is about 89% lower than current LoRaWAN and about 22% lower than 24-bit DevNonce. In conclusion, the proposed scheme is the least likely scheme among all the DevNonce-based schemes proposed so far that the normal node is accidentally mistaken as a replay attacker.

Table 1. Probability that DevNonce duplication occurs within X years

	1 year	**3 years**	**5 years**	**10 years**
Current LoRaWAN	0.6337 (*K*=365)	0.9950 (*K*=1095)	0.9951 (*K*=1825)	0.9951 (*K*=3650)
24-bit DevNonce	0.0039 (*K*=365)	0.0350 (*K*=1095)	0.0943 (*K*=1825)	0.3274 (*K*=3650)
Proposed Initial Join	0.0008 (*K*=12)	0.0090 (*K*=36)	0.0258 (*K*=60)	0.1015 (*K*=120)

5.2 Experiment

To verify that the calculated results for the proposed initial join are valid, we conducted an experiment to generate a 16-bit DevNonce in real environment using the SK-IM880B demo board. This experiment is based on the LoRaWAN end node source code provided by Semtech [8]. In particular, the experimental algorithm was implemented by modifying the random() function in the sx1272.c file used to create DevNonce. The main purpose of our experiment is to figure out how many attempts are needed before DevNonce duplication occurs on the real device. At first, we create a DevNonce on the end node. Then compare this value with the previous generated DevNonces. If there is no identical value, save the newly generated DevNonce in the list and repeat the same operation. If the same value exists, stop the process and see how many DevNonces have been created so far. We repeated this experiment about 30,000 times. Based on the experimental results, we calculated the probability that duplication of 16-bit DevNonce occurs in K join attempts and Fig. 5. shows the probability. With this Fig. 5., we can confirm that our experiment support the formula-based calculation in the previous section.

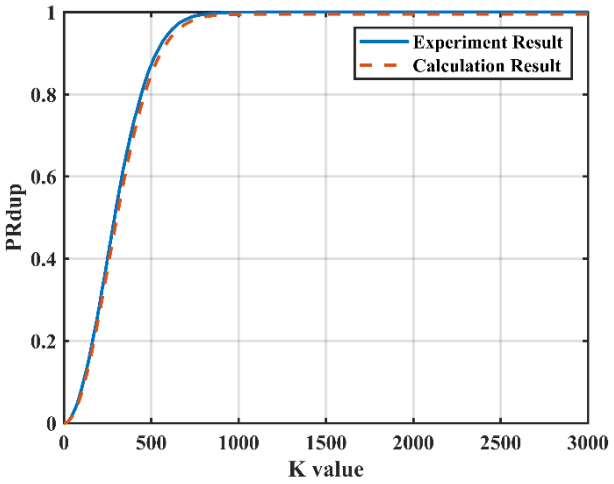

Figure 5. Probability that 16-bit DevNonce duplication occurs with K attempts – Experiment and Calculation Result

6. Conclusion

In this paper, we propose a new LoRaWAN replay attack prevention scheme. In the proposed scheme, we newly defined the initial and non-initial join request. The non-initial join request is used in normal situations, and checks the validity of NwkSKey to prevent replay attacks. The initial join request is used only in exceptional cases where the NwkSKey does not exist in the end node, and prevents replay attack using the DevNonce. With this initial join request, our scheme works well even if the NwkSKey is lost on the end node. Also, our scheme is compatible with the current LoRaWAN because it does not require any packet structure change and there is no additional overhead. For the probability of DevNonce duplication, our scheme shows about 60-89% lower probability than current LoRaWAN and 0.03 ~ 22% lower than 24-bit DevNonce. Real-world experiments using the actual LoRaWAN device also support these results.

7. ACKNOWLEDGMENTS

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(NRF-2015R1D1A1A01058928).

8. REFERENCES

[1] Raza, U., Kulkarni, P., and Sooriyabandara, M. 2017. Low power wide area networks: An overview. *IEEE Surveys Communications & Tutorials.* 19, 2 (Jan. 2017), 855-873. DOI= https://doi.org/10.1109/COMST.2017.2652320.

[2] Georgiou, O. and Raza, U. 2017. Low Power Wide Area Network Analysis: Can LoRa Scale?. *IEEE Wireless Communications Letters.* 6, 2 (Apr. 2017), 162-165. DOI= https://doi.org/10.1109/LWC.2016.2647247.

[3] LoRa Alliance. 2016. LoRaWAN™ Specification 1.0.2. (July 2016).

[4] Zulian, S. 2016. *Security threat analysis and countermeasures for LoRaWAN™ join procedure.* Master's thesis. University of Padova, Padova, Italy.

[5] Tomasin, S., Zulian, S., and Vangelista, L. 2017. Security Analysis of LoRaWAN Join Procedure for Internet of Things Networks. In *Wireless Communications and Networking Conference Workshops* (The San Francisco, The USA, March 19 - 22, 2017). WCNCW '17. IEEE, 1-6. DOI=https://doi.org/10.1109/WCNCW.2017.7919091.

[6] Na, S., Hwang, D., Shin, W., and Kim, K. 2017. Scenario and Countermeasure for Replay Attack Using Join Request Messages in LoRaWAN. In *International Conference on Information Networking* (The Da Nang, The Vietnam, January 11 - 13, 2017). ICOIN '17. IEEE, 718-720. DOI= https://doi.org/10.1109/ICOIN.2017.7899580.

[7] LoRa Alliance. 2016. NB-IoT vs LoRa™ : Which could take gold? (September 2016). Retrieved from https://docs.wixstatic.com/ugd/eccc1a_fb4ea35d44a5492b8c7c58c3b64cdc3d.pdf.

[8] Semtech. 2013. LoRaWAN endpoint stack implementation and example projects. Retrieved from https://github.com/Lora-net/LoRaMac-node.