Hindawi
Wireless Communications and Mobile Computing
Volume 2017, Article ID 6590713, 12 pages
https://doi.org/10.1155/2017/6590713

Research Article

A Dual Key-Based Activation Scheme for Secure LoRaWAN

Jaehyu Kim and JooSeok Song

Yonsei University, 3rd Engineering Building C505, 50 Yonsei-ro, Seodaemun-gu, Seoul 03722, Republic of Korea

Correspondence should be addressed to Jaehyu Kim; jaehyu_kim@yonsei.ac.kr

Received 28 April 2017; Accepted 12 October 2017; Published 6 November 2017

Academic Editor: Haiyu Huang

Copyright © 2017 Jaehyu Kim and JooSeok Song. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

With the advent of the Internet of Things (IoT) era, we are experiencing rapid technological progress. Billions of devices are connected to each other, and our homes, cities, hospitals, and schools are getting smarter and smarter. However, to realize the IoT, several challenging issues such as connecting resource-constrained devices to the Internet must be resolved. Recently introduced Low Power Wide Area Network (LPWAN) technologies have been devised to resolve this issue. Among many LPWAN candidates, the Long Range (LoRa) is one of the most promising technologies. The Long Range Wide Area Network (LoRaWAN) is a communication protocol for LoRa that provides basic security mechanisms. However, some security loopholes exist in LoRaWAN's key update and session key generation. In this paper, we propose a dual key-based activation scheme for LoRaWAN. It resolves the problem of key updates not being fully supported. In addition, our scheme facilitates each layer in generating its own session key directly, which ensures the independence of all layers. Real-world experimental results compared with the original scheme show that the proposed scheme is totally feasible in terms of delay and battery consumption.

1. Introduction

Today, we are living in the Internet of Things (IoT) era where billions of IoT devices are deployed all over the world. According to a report from Ericsson [1], the number of connected IoT devices will reach 28 billion by 2021. These devices produce a massive amount of data and transfer the information to cloud servers, which can be accessed anytime and anywhere. Revolutionary changes are brought into our lives.

Many approaches have been taken to realize various types of communication used in the IoT environment. In the last few years, short-range communication technologies, such as Bluetooth, ZigBee, and Z-Wave, have been popular for utilizing resource-constrained IoT devices because of their low energy consumption [2]. However, their short communication range makes them difficult to use for important IoT applications that require a wide communication range, such as smart city [3]. Although cellular networks provide a wide coverage area, they are also not fully suitable for the IoT environment because of its complexity and cost [4]. To complement the shortcomings of these conventional approaches, Low Power Wide Area Networks (LPWAN) technologies have recently been developed. They are devised to enable long range communication with low battery consumption. With these technologies, even resource-constrained small sensors or actuators can send messages up to tens of kilometers and survive for several years even without a power source [5].

Among the recently proposed LPWAN technologies, such as SigFox, LoRa, Weightless, Ingenu, and Telensa, LoRa is one of the most competitive technologies because of its low power consumption and low cost design [6]. LoRa is a physical layer protocol that enables low power and long-distance communication up to 15 km using chirp spreading spectrum modulation [4]. LoRaWAN is an upper layer protocol based on LoRa that defines the structure and operation of the entire system [7]. LoRaWAN's asynchronous communication scheme enables much longer battery lifetime by reducing the overhead caused by synchronization [5].

While many LPWAN technologies are primarily focused on issues such as battery consumption and communication range, security is also an important issue. In the IoT environment, the importance of security becomes much greater than ever before. The IoT can be a big threat to privacy because it is

closely related to a user's real life. Moreover, the damage from security incidents can be unprecedentedly enormous due to the large scale and connectivity of the IoT environment. To prepare for this situation, previous studies on IoT security [8–10] have addressed some important factors, one of which is key management. According to the research, cryptographic keys can be leaked through various attacks, considering that IoT sensing devices are usually deployed where the attacker can access them. This can be applied to LoRaWAN as well. LoRaWAN specifications [7] emphasize that the key must be uniquely managed to minimize the damage caused by key leakage. This means that when the key is extracted from an end node, it should not affect the other nodes. However, it is not enough and problems still occur with key updates. Although LoRaWAN uses cryptographic keys for several security mechanisms, such as authentication, encryption, and integrity checking, the current LoRaWAN specifications provide update of these keys partially. In some cases, an end node may have to keep using certain keys without changing them during its lifetime. Thus, at some point in the future, if the key is leaked, all the data that the end node has transferred may be passed on to the attacker. To prepare for the attack, keys must be updated periodically, as pointed out in many previous studies [11–14]. How the session key is created in the current LoRaWAN is also problematic. As depicted in Figure 1, each session key is used in a different layer. Thus, the current way in which both session keys are only created by a network server can violate the independence between layers. According to [15], this system could lead to a conflict of interest between the network server and the application server.

In this paper, we propose a dual key-based activation scheme with a new key called a network key (NwkKey). Our scheme resolves the problem that key updates are not available in some cases. We also redefine the operation of each server in the key generation process. In our scheme, a network server and an application server generate a network session key (Nwk_SKey) and an application session key (App_SKey), respectively, so that each layer works completely independently. Moreover, our scheme does not require any additional entities such as a trusted third party. Finally, we demonstrate the feasibility of our scheme through a real-world test. To the best of our knowledge, this is the first attempt to improve the security of LoRaWAN activation.

The rest of this paper is organized as follows. Section 2 provides related works. Section 3 is about LoRaWAN architecture. In Section 4, we provide basic information about LoRaWAN end node activation. In Section 5, a detailed explanation of our proposed scheme is provided. Section 6 is a security analysis of our scheme. In Section 7, we evaluate the performance of our scheme. Section 8 provides our conclusion about this research.

2. Related Works

A security report [16] written by Miller of MWR Infosecurity provides LoRaWAN's possible vulnerabilities and countermeasures as well as basic description of LoRaWAN security. According to the report, all LoRaWAN entities should be prepared for vulnerabilities that can occur during key management, communications, and Internet connection. Especially in case of an end node, the report emphasizes that even if cryptographic keys are leaked through side channel attacks, this should not affect other parts of the system.

In [15], Girard of Gemalto pointed out a problem with LoRaWAN's key provisioning method. In the current LoRaWAN, the network server generates both session keys. This means that the network server generates even the application session key to be used by the application server. According to [15], this could lead to a conflict of interest between the network server and the application server. As a solution to this problem, the author proposes a new LoRaWAN network structure with the trusted third party.

In [17], Zulian analyzed the DevNonce of LoRaWAN. The DevNonce is a random number generated by the end node. It is used for replay attack prevention as well as session key generation. Replay attack prevention works in such a way that the network server determines an invalid message by checking whether previously used DevNonce is contained or not. The author mathematically analyzed the method and determined that the end node can be unavailable with a certain probability under the current DevNonce system. To alleviate this problem, the author proposed increasing the size of the DevNonce field to 24 or 32 bits.

Naoui et al. proposed a new security architecture for LoRaWAN [18]. Their scheme uses the concept of a proxy node, which performs several other functions, including the basic function of the conventional LoRaWAN gateway. In particular, proxy nodes evaluate each other's trustworthiness to create a table and forward it to the end node. The end node can then communicate through the proxy node that has the highest trust value.

The current LoRaWAN has problems with key update and session key generation. In the case of the key update, it has not been addressed in any LoRaWAN security study, despite its seriousness. The solution proposed by [15] to solve the problem of session key generation also has some disadvantages. Because of the newly added trusted third party, the whole join procedure becomes more complex and communication overhead is increased. It is also difficult to be applied to the existing LoRaWAN network already deployed without the trusted third party. In this paper, we propose a dual key-based activation scheme that fully supports the key update and resolves the problem of session key generation without any additional entities.

3. LoRaWAN Architecture

In this section, we briefly describe the architecture of the LoRaWAN network and its entities. We also provide a description of the protocol architecture and message format that are used in the LoRaWAN network environment.

3.1. LoRaWAN Network Architecture. As shown in Figure 1, the LoRaWAN network uses a star topology in which an end node can send messages to multiple gateways that communicate with the network server. Since an end node does not belong to a specific gateway, more than one gateway

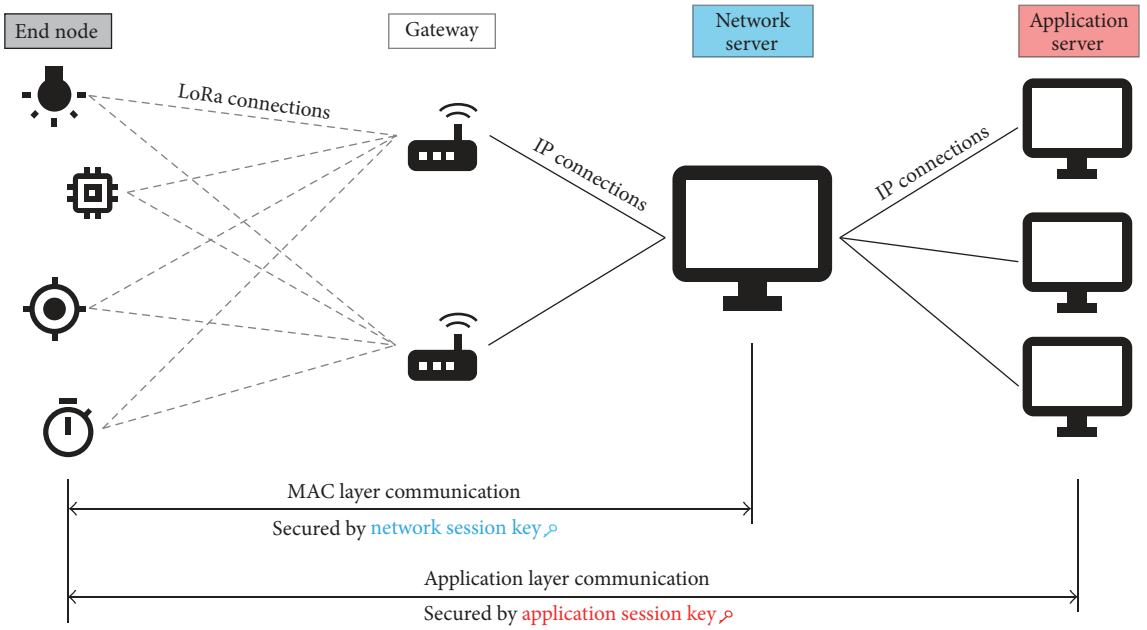


Figure 1: LoRaWAN network structure.

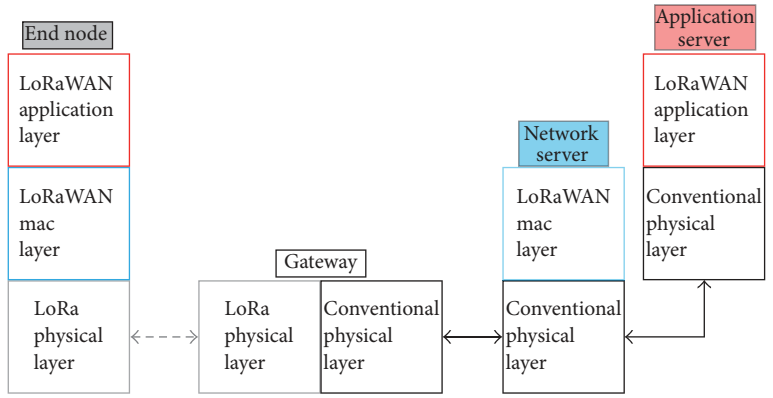


Figure 2: LoRaWAN protocol architecture.

can receive a message sent by an end node [19]. LoRa radio technology is used in communications between an end node and the gateways. The gateways and network server are connected via standard IP connections. The following is a brief description of the entities defined in the LoRaWAN specifications [7].

(i) End node: a LoRaWAN end node is typically used to send small amounts of data at low frequencies over long distances. It can be utilized in various fields such as smart city, smart building, factory automation, farm automation, and logistics.

(ii) Gateway: a LoRaWAN gateway receives packets from the end node via a LoRa radio link. It then forwards them to the network server through the IP connection.

(iii) Network server: the LoRaWAN network server manages the entire network. When it receives packets, it removes the redundancy of packets and performs a security check and then determines the most suitable gateway to send back an acknowledgement message.

3.2. LoRaWAN Protocol Architecture. Figure 2 shows the protocol architecture of LoRaWAN. As shown in this figure, LoRaWAN's protocol consists of a MAC layer and an application layer, and it operates based on the LoRa physical layer. The packet format is displayed in Figure 3. The maximum payload lengths M and N vary with the data rate. It is specified in [20].

(i) MAC layer: the packet processed in the MAC layer consists of a MAC Header (MHDR), a MAC Payload, and a Message Integrity Code (MIC). In a join procedure for end node activation, the MAC Payload can be replaced by join request or join accept messages. The entire MAC Header and MAC Payload portion is used to compute the MIC value with a network session key

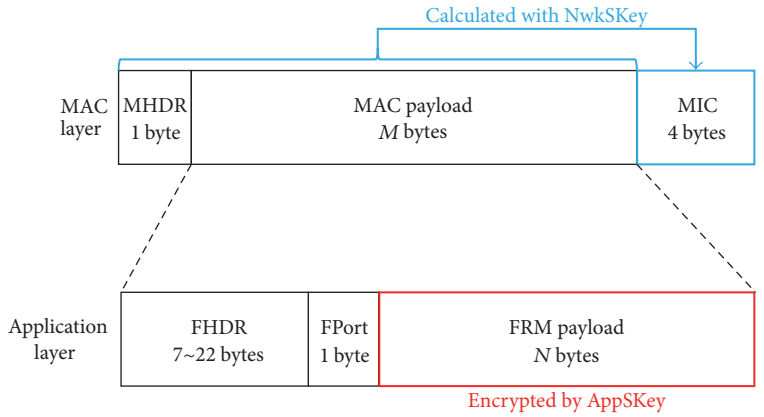


Figure 3: LoRaWAN packet format.

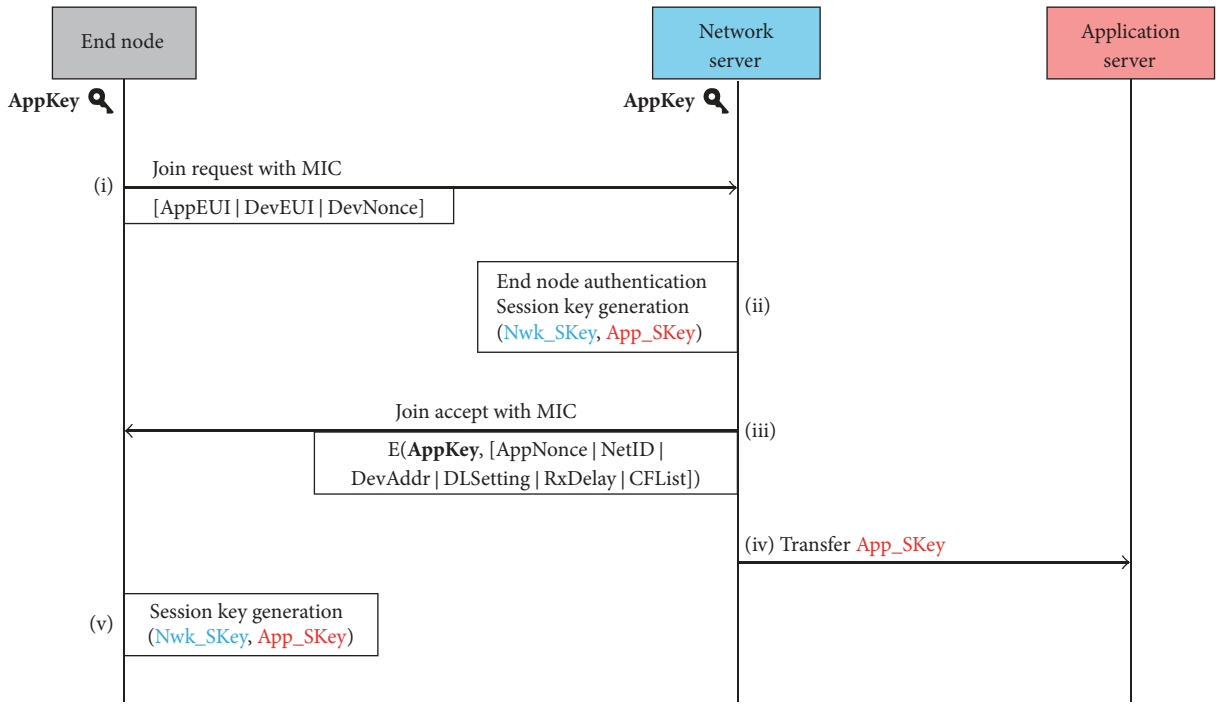


Figure 4: LoRaWAN join procedure.

(Nwk_SKey). The MIC value is used to prevent the forgery of messages and authenticate the end node.

(ii) Application layer: the MAC Payload handled by the application layer consists of a FRM Header (FHDR), an FPort, and a FRM Payload. The FPort value is determined depending on the application type. The FRM Payload value is encrypted with an application session key (App_SKey). This encryption is based on the AES_128 algorithm.

4. LoRaWAN End Node Activation

When a new end node is added to a LoRa network, it should go through an activation process. Through the activation process, both session keys are shared between the end node and the network server. Currently, LoRaWAN provides two types of activation methods. One is over-the-air activation (OTAA) and the other is activation by personalization (ABP).

4.1. Over-the-Air Activation. In the OTAA mode, an end node communicates with the network server to perform the activation process, which is called join procedure. According to the LoRaWAN specifications [7], the OTAA mode is used when an end node is deployed or reset. Figure 4 shows the LoRaWAN join procedure. A detailed explanation of each step is as follows.

(i) Join request message: by sending a join request message, the end node starts the join procedure.

DevEUI, AppEUI, and DevNonce are included in the join request. DevEUI and AppEUI refer to the global end node and application identifier, respectively. They follow the IEEE EUI-64 address space format. The DevNonce is a random number generated by the end node. The MIC value of join request is calculated by the following formula:

$$\begin{aligned} cmac &= aes128\_cmac\left(AppKey, MHDR \mid AppEUI \mid DevEUI \mid DevNonce\right) \\ \mathbf{MIC} &= cmac\left[0 \cdots 3\right]. \end{aligned} \tag{1}$$

An application key (AppKey) is preshared between the end node and the network server.

(ii) After the network server receives the join request, it performs the replay attack prevention process, which is based on the DevNonce. If the DevNonce in the join request is previously used, the network server determines that the message is invalid and that the join process will fail. If the message is valid, the network server authenticates the end node with the MIC value. If the end node passes the authentication, the network server generates an Nwk_SKey and an App_SKey by the following formula:

$$\begin{aligned} Nwk\_SKey &= aes128\_encrypt\left(AppKey, \texttt{0x01} \mid AppNonce \mid NetID \mid DevNonce \mid pad_{16}\right) \\ App\_SKey &= aes128\_encrypt\left(AppKey, \texttt{0x02} \mid AppNonce \mid NetID \mid DevNonce \mid pad_{16}\right). \end{aligned} \tag{2}$$

AppNonce is a random number generated by the network server. NetID is a 24-bit field. Its 5 LSBs are called NwkID which is used to separate addresses of geographically duplicated LoRa networks. The other bits of NetID can be freely determined by the network server.

(iii) Join accept message: a join accept message contains AppNonce, NetID, DevAddr, DLSettings, RxDelay, and CFList. The DevAddr is a 32-bit identifier of the end node within the current network. The 7 MSBs of DevAddr are referred to as the NwkID, which is also contained in NetID. The other bits can be arbitrarily chosen by the network server. DLSettings contains several values related to the downlink configuration. RxDelay is a delay between the transmission and reception process. CFList is an optional field that is about channel frequencies. Finally, the whole join accept message is encrypted with the AppKey.

(iv) Transfer App_SKey: since the App_SKey is devised to secure end-to-end communications between the end node and the application server, it should be transferred from the network server to the application server. The LoRaWAN specification does not specify when and how to exchange App_SKey with the application server. We thought it is an essential part and so included it in the join procedure.

(v) After receiving the join accept message, the end node decrypts it and generates session keys using extracted parameters.

4.2. Activation by Personalization. ABP is the way in which an end node can belong to a particular LoRa network without performing a join procedure under certain circumstances. In the ABP mode, the end node does not have DevEUI, AppEUI, and AppKey, which are essential for join procedure. Instead, both session keys required for LoRaWAN communications and DevAddr are preloaded on the end node.

4.3. Problem Statement

(1) OTAA key update: in the OTAA mode, authentication and session key agreement is performed using the AppKey preshared between the end node and the network server. In this process, one of the most critical problems is that updating the AppKey is not supported by the LoRaWAN specifications. Under the current standard, session keys can be updated several times, but the AppKey that is used to generate them cannot be updated. In other words, the end node has to use only one AppKey for a lifetime. As pointed out in several previous IoT security studies [8–10], we have to prepare for key leakage, which can have various causes, such as node capture attacks and side channel attacks.

In this respect, LoRaWAN AppKey, which cannot be updated, can cause serious security problems. If the AppKey is leaked by an attacker, the attacker can get the contents of all join accept messages that have been sent up to that point. As shown in (2), AppNonce, NetID, and DevNonce are used to generate session keys. Among them, the AppNonce and NetID are contained in the join accept message. The DevNonce can easily be obtained in the join request transmitted without encryption. By using these parameters, the attacker can restore all the session keys used in the past. Thus, the attacker can steal all the data that the target node had previously transmitted. From this perspective, many previous studies on key management [11–13] and NIST [14] have emphasized that the key must be updated periodically.

(2) ABP key update: in the ABP mode, the AppKey is not preloaded on the end node. Since the AppKey is essential to the join procedure, the end node cannot perform it, which means that there is no way of updating session keys. Therefore, in the ABP mode, the end node must use the same session key throughout its lifetime. This can also pose a similar security threat to the end node, as discussed in (1). If the attacker successfully steals these keys, he or she can get all the data sent from the target node that were

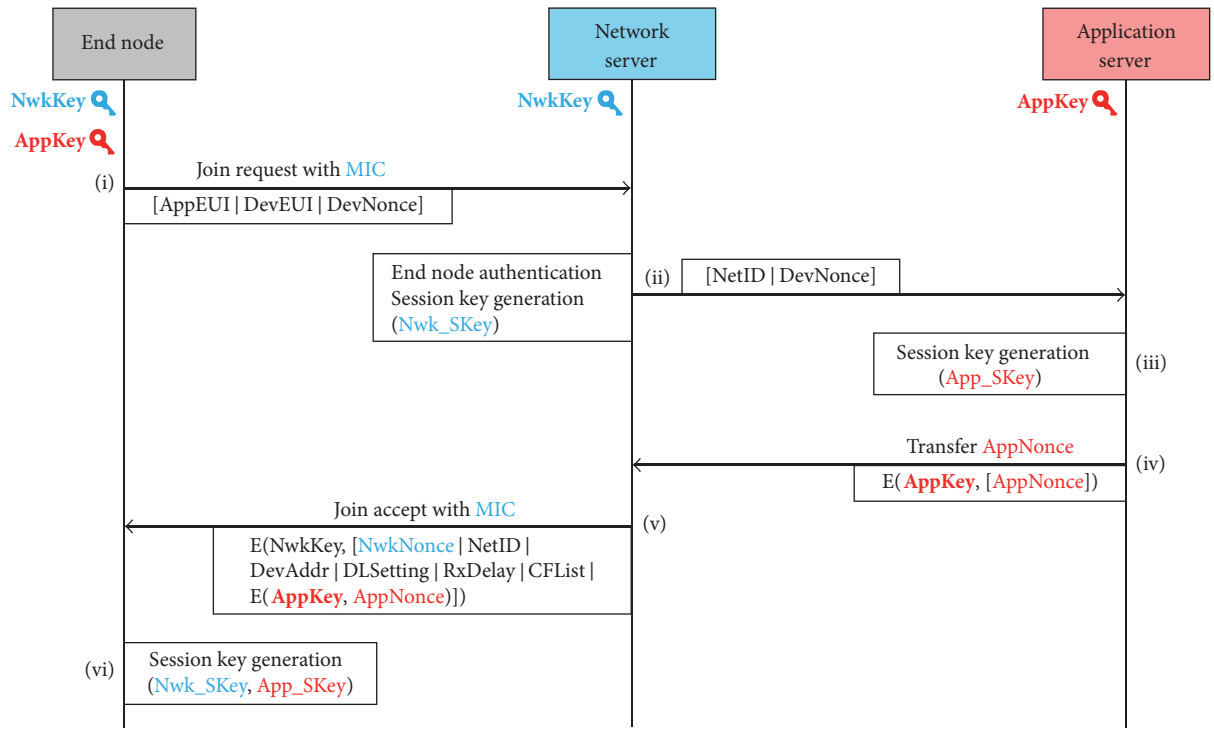


Figure 5: Dual key join procedure.

protected by the session keys. This is why supporting the key update in the ABP mode is an important issue.

(3) Session key generation: under the current LoRaWAN session key generation system, the network server generates all the session keys alone. Since the Nwk_SKey and the App_SKey are used in different layers, this system does not guarantee independence between layers. According to [15], there is even the risk that the network server with the App_SKey can intercept the application layer data. Thus, we need to construct a new system in which each layer independently generates its own session key.

5. Dual Key-Based End Node Activation

5.1. Dual Key-Based Over-the-Air Activation. Compared to the original join procedure, the notable feature of our scheme is the existence of the NwkKey. The NwkKey has the same properties as the AppKey. They are of the same length and should not be deduced from the public information of the end node. They should not also be shared with other end nodes. In our scheme, the NwkKey and the AppKey are preloaded together on the end node. The NwkKey is shared with the network server, and the AppKey is with the application server. During the proposed join procedure, Nwk_SKey and App_SKey are generated from the NwkKey and the AppKey, respectively.

Our scheme works in two modes, initial and noninitial.

(1) Initial Join Procedure. The initial mode is applied when an end node performs the join procedure for the first time. Figure 5 represents the proposed initial join procedure, and the details are as follows.

(i) Join request: the join request message is created in the same manner as the original scheme. The message includes AppEUI, DevEUI, and DevNonce. The MIC is also calculated in the same way, except that the NwkKey is used instead of the AppKey. In our scheme, preshared key between the end node and the network server is not the AppKey but the NwkKey.

$$cmac = aes128\_cmac\left(\mathbf{NwkKey}, MHDR \mid AppEUI \mid DevEUI \mid DevNonce\right) \quad (3)$$

$$\mathbf{MIC} = cmac\left[0 \cdots 3\right].$$

(ii) On receipt of the join request, the network server authenticates the end node by recalculating the MIC value. Since the end node and the network server share the NwkKey, the end node can be authenticated. If the message is valid, the network server generates the Nwk_SKey with the NwkKey and transfers NetID and DevNonce to the application server.

$$Nwk\_SKey = aes128\_encrypt\left(\mathbf{NwkKey}, \texttt{0x01} \mid \mathbf{NwkNonce} \mid NetID \mid DevNonce \mid pad_{16}\right). \quad (4)$$

Compared to the original join procedure, the NwkNonce and the NwkKey are used instead of the AppKey and the AppNonce. The NwkNonce

is a random number that has essentially the same properties as the AppNonce.

(iii) Application server generates an App_SKey after receiving the NetID and DevNonce from the network server. The generation method is as follows:

$$AppSKey = aes128\_encrypt\left(\mathbf{AppKey}, \texttt{0x01} \mid \mathbf{AppNonce} \mid NetID \mid DevNonce \mid pad_{16}\right). \quad (5)$$

(iv) When the App_SKey generation is completed, the application server sends the AppNonce to the network server. At this time, the AppNonce is encrypted with the AppKey. Since the network server does not have the AppKey, it cannot decrypt the ciphertext.

(v) Join accept message: after receiving the encrypted AppNonce from the application server, the network server sends a join accept message. In the proposed scheme, the NwkNonce and the encrypted AppNonce are included. The rest is the same as the original join accept message. The entire message is encrypted with the NwkKey before transmission.

(vi) The end node decrypts the join accept message. After that, it generates the Nwk_SKey and the App_SKey with the extracted parameters.

When communication is initiated with the newly created session keys, the end node and both servers immediately discard the NwkKey and the AppKey. This is to prevent the key from being leaked by the attacker in the future.

(2) Noninitial Join Procedure. If the end node already joined to the network through the initial join procedure needs to perform join procedure again, the noninitial join procedure is performed. The noninitial mode is almost the same as the initial mode, except that the session keys created in the previous join procedure are used instead of the NwkKey and the AppKey. In other words, the noninitial join procedure is the process of creating new session keys from the old session keys. The joined end node no longer has the NwkKey and the AppKey. Therefore, subsequent join procedures are performed in the noninitial mode. Through this process, the end node can update the keys used for LoRaWAN's security mechanisms.

5.2. Dual Key-Based Activation by Personalization. In the current LoRaWAN's ABP mode, both session keys and DevAddr are directly mounted on the end node. Since there are no DevEUI, AppEUI, and AppKey, the join procedure cannot be performed. This means that the session keys mounted on the end node cannot be automatically updated. For the absence of an AppKey, our proposed noninitial join procedure that utilizes both session keys can be a solution. However, a problem still remains, in that the join request cannot be made due to the absence of DevEUI and AppEUI. Therefore, we propose a new join request for ABP mode as follows:

$$JoinReqestforABP = [DevAddr \mid DevNonce]. \quad (6)$$

It uses DevAddr as an identifier. All the other steps, such as session key generation and join accept processing, can be done in the same manner as the noninitial join procedure. As a result, the end node activated via ABP mode can also update its session keys.

6. Security Analysis

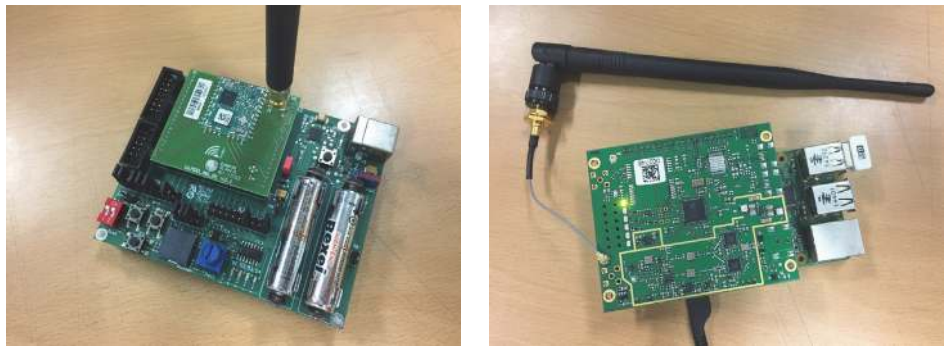
6.1. Basic Security Mechanisms. Our scheme satisfies the same security requirements as LoRaWAN, such as authentication, message integrity, data confidentiality, and replay attack prevention. End node authentication is achieved by using the NwkKey and the MIC value. When a join request arrives, the network server authenticates the end node by recalculating the MIC value with the NwkKey. The MIC is also used for message integrity checking. If the recalculated MIC value is different from the transmitted one, this means that the message is manipulated by unauthorized entities. Application data is encrypted by the App_SKey. In the current LoRaWAN, since the network server generates the App_SKey, application data confidentiality may not be perfectly guaranteed. However, in the proposed scheme, the App_SKey is only shared between the end node and the application server. Thus, application data confidentiality is guaranteed. Replay attack prevention is provided by the DevNonce. When a previously used DevNonce is contained in a join request, the network server considers it invalid.

6.2. Key Update. In the current LoRaWAN activation process, the key update is not fully supported. In the OTAA mode, the end node cannot update the preloaded AppKey and in the ABP mode, preloaded session keys cannot be updated. The end node must use these keys for a lifetime without updating. Thus, if the key is stolen by the attacker, he or she can steal all the data that the target node had previously transmitted.

On the contrary, in our scheme, the end node can update keys in any cases. Regardless of the activation mode in which the end node is activated, the end node can update the key using the proposed initial or noninitial join procedure. Another important aspect of our scheme is that once the key is updated, the previously used key is discarded. In case of the initial join, preloaded NwkKey and AppKey are discarded after the procedure is done. Through the noninitial join, both previously used session keys are discarded. All keys are valid only for the session in our scheme. Thus, the data transmitted in the previous sessions can be protected even if the current session keys are leaked.

6.3. Session Key Generation. LoRaWAN network communication consists of two layers. Each layer has a different session key. Under the current LoRaWAN session key generation system, the network server creates both session keys. Therefore, the security mechanism of each layer is not completely isolated. The network server with the App_SKey can access the application layer data, which should not be permitted.

However, in our scheme, each layer generates its own key. The network server creates the Nwk_SKey, and the application server creates the App_SKey. This means that the network server is no longer involved in creating App_Skey



(a) End node: SK-IM880B demo board (b) Gateway: Raspberry Pi2 with IMST IC880A board

Figure 6: LoRaWAN devices.

and cannot access the application layer data. Thus, compared with the previous method, each layer independently operates a security mechanism.

7. Performance Evaluation

We performed a real-world experiment to demonstrate the feasibility of the proposed scheme. In this section, we provide detailed information on the experimental environment and an analysis of the experimental results.

7.1. Experimentation Environment

(1) Hardware Environment. We have installed a private LoRaWAN network consisting of four entities: the end node, gateway, network server, and application server. Figure 6 shows LoRaWAN devices that we used in this experiment. Hardware specification is shown in Table 1.

(i) End node: we installed the end node by uploading the source code [21] provided by Semtech to the demo board included in SK-IM880B [22].

(ii) Gateway: we made a gateway device by connecting the Raspberry Pi 2 and IMST IC880A [23] board according to the tutorial [24, 25] provided by Semtech and The Things Network. We uploaded the LoRaWAN gateway source code [26, 27] provided by Semtech to complete the gateway installation.

(iii) Network server: there are open source projects for implementing LoRa network server [28, 29]. We established the network server by installing these source codes on Ubuntu OS.

(iv) Application server: there is also an open source project for LoRa application server [30]. It is installed on our application server computer, which runs on Ubuntu OS.

We implemented the proposed scheme by modifying the source code on each entity.

(2) Network Environment. We installed the network environment according to [20]. In this document, parameters related to LoRa transmission, such as default channels, frequency, date rate, and delays, are specified.

Table 1: Hardware specification.

End node	STM32L151CB MCU, 128K Flash, 10K RAM, IM880B-L Module
Gateway	Raspberry Pi 2 with IC880A, Wi-Fi connection
Network server	Intel Core i5-470UM 1.33 GHz CPU, 4 G RAM, Ethernet connection
Application server	Intel Core 2 6600 2.4 GHz CPU, 4 G RAM, Ethernet connection

(i) Band: in South Korea where we have conducted experiments, the LoRa dedicated band is 920–923 MHz [20]. Currently, however, student researchers face difficulty in obtaining experimental equipment for the band. Therefore, we conducted experiments with EU 863–870 MHz band equipment, which can be easily purchased online. Although the band is being used for other purposes in South Korea, we determined that simply verifying the feasibility of our scheme is possible.

(ii) Reception windows: Figure 7 shows when reception windows open in an end node. After completion of the join request transmission, the end node opens two reception windows. A join accept packet can be received only when the reception window is open. The first reception window (RX1) is opened Join Accept Delay1 seconds after the completion of the join request transmission. The Join Accept Delay1 is defined as 5 seconds for EU band. By default, the join accept packet received by RX1 uses the same frequency and same date rate as the join request. The second reception window (RX2) opens after the Join Accept Delay2 and uses the predefined channel. The Join Accept Delay2 is defined as 6 seconds for EU band. The network server decides which reception window to use when creating a join accept message. Since the predefined channel for RX2 is not contained

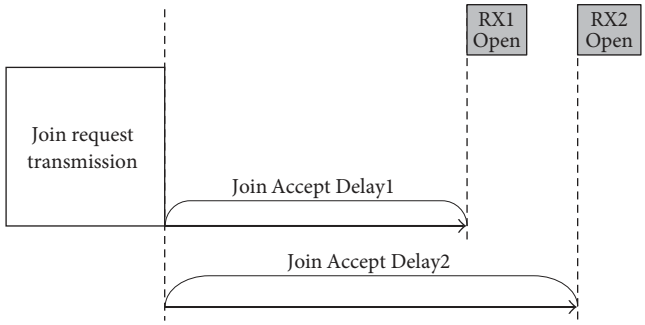


Figure 7: LoRaWAN reception windows.

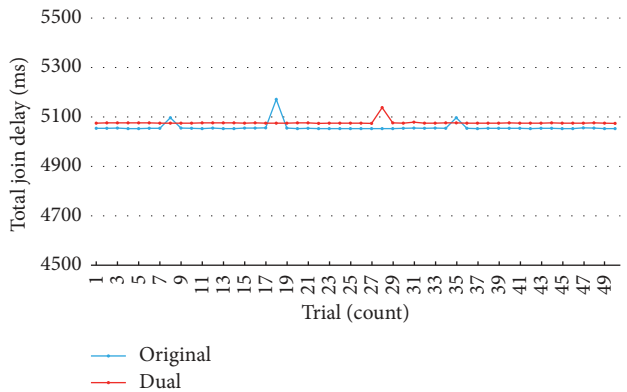


Figure 8: Total join delay.

in the default channel list provided in [20], we use RX1 as a default setting in our experiment.

7.2. Experimentation Results. In this section, we provide the experimentation results. Our proposed scheme has additional work on the end node and server-side compared to the original scheme. This may increase delay and battery consumption. Thus, we compared with the proposed initial join procedure and the original join procedure in terms of delay and battery consumption, which are key elements of feasibility.

(1) Delay. At first, we measured the total join delay from the end of the join request transmission until the end of the join procedure on the end node. Figure 8 shows the experimental result after performing the original join procedure and the proposed initial join procedure 50 times, respectively. Some of the abnormal values seemed to be caused by the temporarily impaired LoRa wireless link. Table 2 shows the average time spent on each scheme. According to the results, the total join delay in the proposed initial join procedure increased by about 18 ms on average. To find out the cause of the increase in delay, we analyzed how the delay occurs at each step of the join procedure. Figure 9 shows the structure of the delay that occurs in the join procedure. Both original

Table 2: Total join delay.

	Total join delay
Original join	5057.92 ms
Proposed initial join	5076.64 ms

and proposed join procedure follow this structure. According to the structure, the total delay can be expressed as follows.

$$\begin{aligned} TotalJoinDelay = {} & ComDelay_{joinreq} \\ & + JoinDelay_{gateway} \\ & + ProcTime_{joinacpt} \\ & + ComDelay_{joinacpt}. \end{aligned} \tag{7}$$

$ComDelay_{joinreq}$ means join request communication delay between the end node and gateway. $JoinDelay_{gateway}$ means gateway join delay. $ProcTime_{joinacpt}$ means the join accept processing time of the end node. $ComDelay_{joinacpt}$ means join accept communication delay between the gateway and end node.

(1) $ComDelay_{joinreq}$: the join request had no difference between the proposed scheme and the original scheme. Therefore, it can be assumed that the same delay occurs.

(2) $JoinDelay_{gateway}$: as depicted in Figure 9, the gateway join delay is the time when the join request packet is received on the gateway until the join accept packet starts sending. This value mainly consists of server-side processing time and gateway waiting time. At the end node, RX1 is opened 5 seconds after the join request is sent, and the join accept message must arrive at this time. As shown in Figure 10 and Table 3, the server-side processing time is less than 1 second. So the gateway must wait until RX1 is open on the end node. If the gateway immediately sends a join accept message without the waiting time, a message cannot be received on the end node where RX1 has not yet been opened. Thus, proper waiting time on the gateway is essential for the successful join procedure. In our experiment, the network server deliberately set the gateway join delay to 5 seconds, which is the same as Join Accept Delay1, to generate a proper waiting time on the gateway. According to this mechanism, the gateway waits until the gateway join delay becomes 5 seconds and then sends the join accept message. As a result, the gateway join delay of both schemes is equally 5 seconds.

(3) $ProcTime_{joinacpt}$: as can be seen in Table 4, the join accept processing time increased by about 0.4 ms on average in the proposed scheme. This is because it performed AES encryption once more. However, since the increased value was very small, the effect on the overall delay was negligible.

(4) $ComDelay_{joinacpt}$: according to our analysis, the other factors constituting the total join delay had little effect

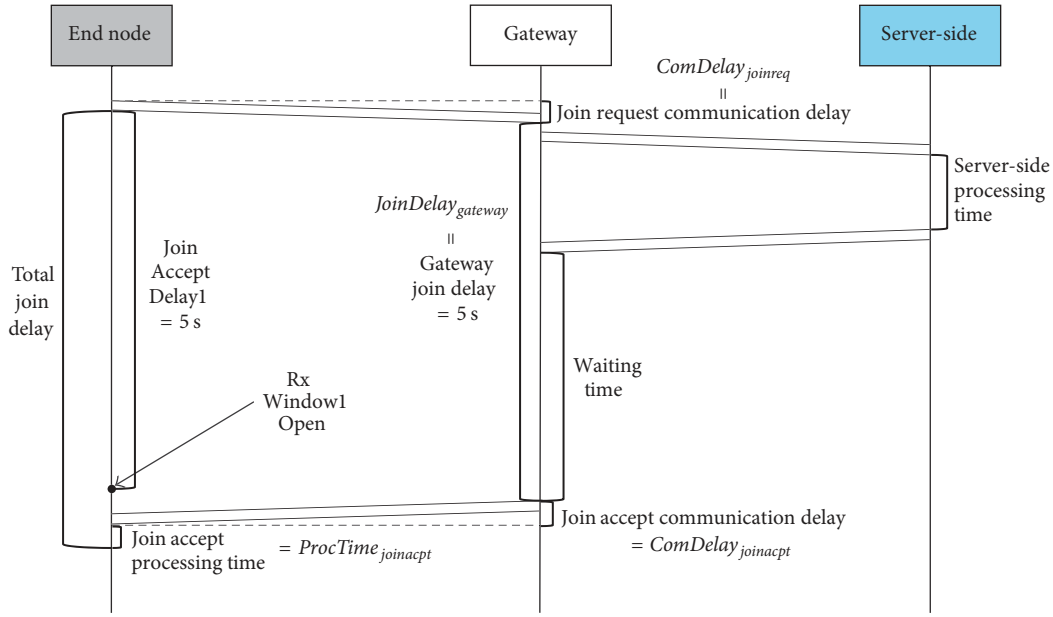


Figure 9: Delay structure.

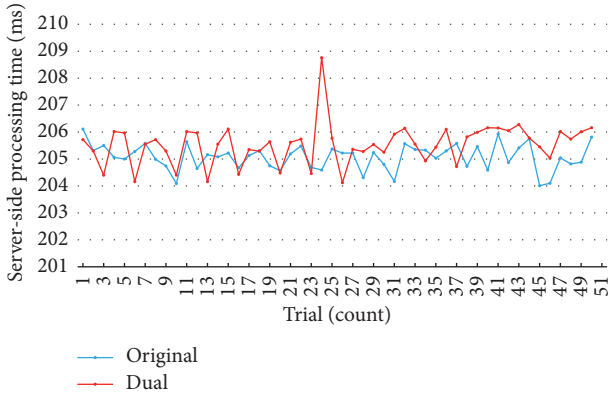


Figure 10: Server-side processing time.

Table 3: Server-side processing time.

	Server-side processing time
Original join	205.07 ms
Proposed initial join	205.53 ms

Table 4: Join accept processing time.

	Join accept processing time
Original join	1.3 ms
Proposed initial join	1.78 ms

on the increased delay of the proposed scheme. Therefore, the join accept communication delay can be inferred as the most decisive factor on the increased delay. The only change in the join accept message of the proposed scheme was the payload length. As shown in Table 5, the payload length of the join accept message increased by 16 bytes. We concluded that this is the main cause of the delay increase.

Table 5: Join accept packet size (without CFList field).

	Join accept payload length
Original join	12 bytes
Proposed initial join	28 bytes

For further analysis, we carried out a simulation with the LoRa Modem Calculator [31] provided by Semtech. The result shown in Figure 11 cannot be directly applied to our experimental results because the calculator does not support the SX1257 transmitter that is equipped in our IC880A gateway board. However, at least we can determine how the transmission time was changed according to the payload length.

We also considered the maximum payload length. According to [20], the maximum payload length varies from 59 bytes to 230 bytes depending on the data rate. Thus, we can ensure that the increased payload length did not affect feasibility because the 28-byte payload length satisfied the maximum payload length in any cases.

As a result, the total join delay of our scheme increased by about 18 ms because the payload length of the join accept packet increased. This payload length completely satisfied the maximum payload length criterion specified by LoRaWAN. Therefore, in terms of delay, our scheme is feasible.

(2) Battery Consumption. Our end node uses two AAA-sized 1.5-V batteries. The source code [21] provided by Semtech has

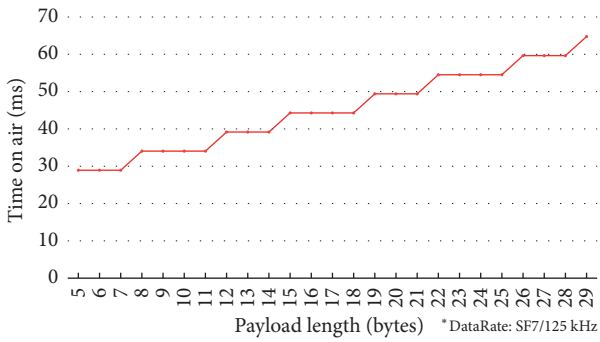


Figure 11: LoRa Modem Calculator result.

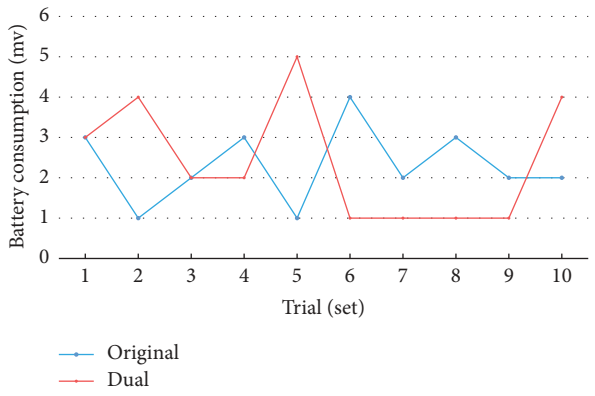


Figure 12: Battery consumption.

Table 6: Average battery consumption.

	Average battery consumption
Original join	0.23 mv
Proposed initial join	0.24 mv

a function that measures the remaining battery capacity of the end node in millivolts. We use this function to measure the battery consumed during the join procedure. However, we cannot obtain meaningful values in one or two join procedures because the battery consumption is less than one millivolt. To overcome the limitation of measurement method, we decide to measure after performing 10 consecutive join procedures. Figure 12 is the result of 10 sets of experiments, 10 times per set.

Due to transmission errors, function errors, and so on, the battery consumption value per set seemed not to be constant. We performed 10 sets, a total of 100 experiments, to ensure that these external factors are equally applied to both schemes. Therefore, the relative battery consumption of the proposed scheme for the original scheme is trustworthy. Table 6 shows that the battery consumption of the proposed scheme is not significantly different from the original scheme. Therefore, in terms of battery consumption, the proposed scheme is feasible.

8. Conclusion

In this paper, we proposed a dual key-based activation scheme. Our scheme uses NwkKey and AppKey to perform the initial join procedure. From the second join procedure, session keys are used, which are created in the previous join procedure. This resolves the key update problem, which was not fully supported in the original scheme. In addition, our scheme makes each layer generate its own session key so that the layers can work independently. We compared the performance of the original scheme and the proposed scheme through a real-world experiment. According to the experimental results, our scheme is feasible in terms of delay and battery consumption.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

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).

References

[1] "Ericsson Mobility Report: On the Pulse of the Networked Society," 2015. http://www.ericsson.com/res/docs/2015/mobility-report/ericsson-mobility-report-nov-2015.pdf.

[2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: a survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[3] X. Xiong, K. Zheng, R. Xu, W. Xiang, and P. Chatzimisios, "Low power wide area machine-to-machine networks: Key techniques and prototype," *IEEE Communications Magazine*, vol. 53, no. 9, pp. 64–71, 2015.

[4] M. Centenaro, L. Vangelista, A. Zanella, and M. Zorzi, "Long-range communications in unlicensed bands: The rising stars in the IoT and smart city scenarios," *IEEE Wireless Communications Magazine*, vol. 230, no. 5, pp. 60–67, 2016.

[5] U. Raza, P. Kulkarni, and M. Sooriyabandara, "Low power wide area networks: an overview," *IEEE Communications Surveys & Tutorials*, 2017.

[6] O. Georgiou and U. Raza, "Low power wide area network analysis: can LoRa scale?" *IEEE Wireless Communications Letters*, vol. 6, no. 2, pp. 162–165, 2017.

[7] N. Sornin, M. Luis, T. Eirich, T. Kramp, and O. Hersent, "LoRaWAN Specification V1.0.2," *LoRa Alliance*, 2016.

[8] Z.-K. Zhang, M. C. Y. Cho, and S. Shieh, "Emerging security threats and countermeasures in IoT," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security (ASIA CCS '15)*, pp. 1–6, ACM, April 2015.

[9] M. M. Hossain, M. Fotouhi, and R. Hasan, "Towards an Analysis of Security Issues, Challenges, and Open Problems in the Internet of Things," in *Proceedings of the IEEE World Congress on Services, SERVICES 2015*, pp. 21–28, July 2015.

[10] K. Zhao and L. Ge, "A survey on the internet of things security," in *Proceedings of the 9th International Conference on Computational Intelligence and Security, CIS 2013*, pp. 663–667, December 2013.

[11] S.-H. Seo, J. Won, S. Sultana, and E. Bertino, "Effective key management in dynamic wireless sensor networks," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 2, pp. 371–383, 2015.

[12] S. Agrawal, R. Roman, ML. Das, A. Mathuria, and J. Lopez, "A novel key update protocol in mobile sensor networks," in *Proceedings of the International Conference on Information Systems Security*, pp. 194–207, Springer, 2012.

[13] J. He, X. Zhang, and Q. Wei, "EDDK: Energy-efficient distributed deterministic key management for wireless sensor networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2011, 2011.

[14] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, *Recommendation for Key Management Part 1: General (Revision 4)*, NIST Special Publication, 2016.

[15] P. Girard, "Low Power Widw Area Networks security," 2015. https://docbox.etsi.org/Workshop/2015/201512_M2MWORKSHOP/S04_WirelessTechnoforIoTandSecurityChallenges/GEMALTO_GIRARD.pdf.

[16] R. Miller, "LoRa Security: Building a Secure LoRa Solution," 2016. https://labs.mwrinfosecurity.com/publications/lo/.

[17] S. Zulian, *Security Threat Analysis and Countermeasures for LoRaWAN Join Procedure*, 2016, http://tesi.cab.unipd.it/53210/.

[18] S. Naoui, M. E. Elhdhili, and L. A. Saidane, "Enhancing the security of the IoT LoraWAN architecture," in *Proceedings of the 5th IFIP International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks, PEMWN 2016*, November 2016.

[19] "LoRaWAN - What is it: A technical overview of LoRa and LoRaWAN," 2015. https://www.lora-alliance.org/portals/0/documents/whitepapers/LoRaWAN101.pdf.

[20] LoRa Alliance Technical committee, LoRaWAN Regional Parameters, 2016.

[21] Semtech, *LoRaWAN endpoint stack implementation and example projects*, 2013. https://github.com/Lora-net/LoRaMac-node.

[22] *SK-iM880B –Long Range Radio Starter Kit* https://wireless-solutions.de/products/starterkits/sk-im880b.html.

[23] *iC880A - LoRaWAN Concentrator 868MHz* https://wireless-solutions.de/products/radiomodules/ic880a.html.

[24] Semtech, *Use with Raspberry Pi*, 2016. https://github.com/Lora-net/packet_forwarder/wiki/Use-with-Raspberry-Pi.

[25] The Things Network, *From zero to LoRaWAN in a weekend*, 2016. https://github.com/ttn-zh/ic880a-gateway/wiki.

[26] Semtech, *LoRa Gateway project* https://github.com/Lora-net/lora_gateway.

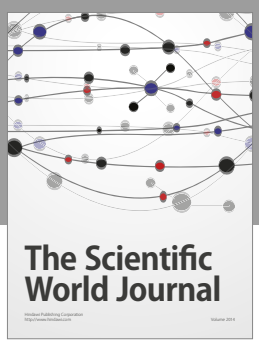
[27] Semtech, *Lora network packet forwarder project* https://github.com/Lora-net/packet_forwarder.

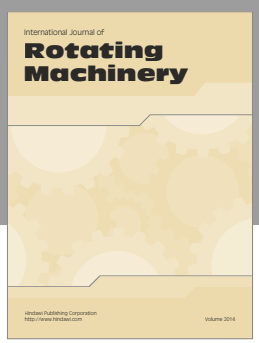
[28] Brocaar, *LoRa Gateway Bridge* https://github.com/brocaar/lora-gateway-bridge.

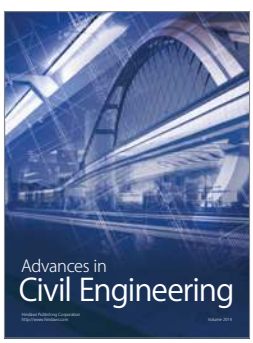
[29] Brocaar, *LoRa Server* https://github.com/brocaar/loraserver.

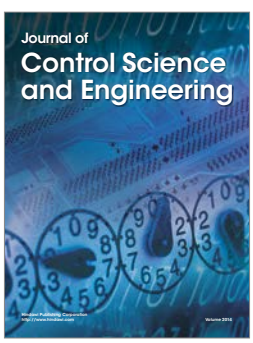
[30] Brocaar, *LoRa App Server* https://github.com/brocaar/lora-app-server.

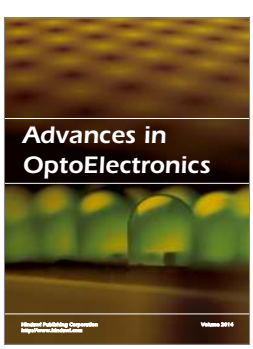
[31] Semtech, *LoRa Calculator* http://www.semtech.com/wireless-rf/rf-transceivers/sx1272/.

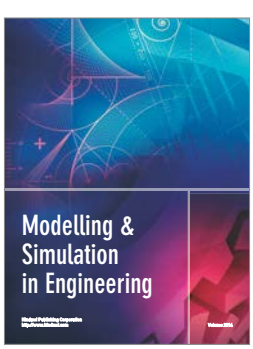


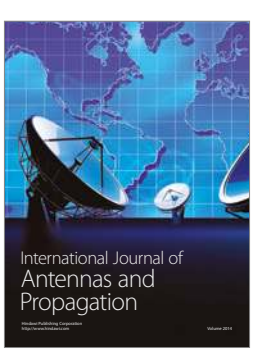


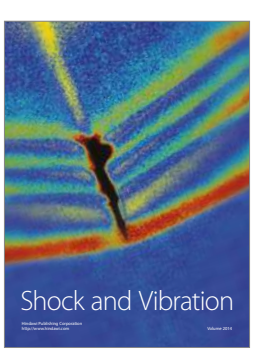


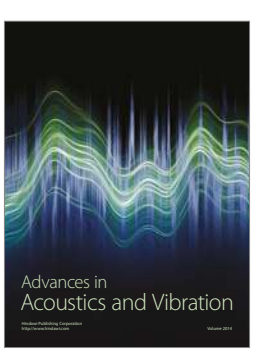












Hindawi

Submit your manuscripts at
https://www.hindawi.com