LoRaWAN Network Server Session Keys Establish Method with the Assistance of Join Server

Kun-Lin Tsai[1(✉)], Fang-Yie Leu[2], Li-Chun Yang[1], Chi Li[1], and Jhih-Yan Liu[1]

[1] Department of Electrical Engineering, Tunghai University, Taichung, Taiwan
kltsai@thu.edu.tw
[2] Department of Computer Science, Tunghai University, Taichung, Taiwan

Abstract. With the development of Internet of Things (IoT), various communication protocols are created to support long range, low cost and low power consumption network environment. The LoRaWAN developed by LoRa Alliance is one of them. The LoRa Alliance Technical Committee proposed LoRaWAN specification version 1.1 to detail message communication structure in each network layer and to enhance network security. The network server in LoRaWAN specification version 1.0 is divided into three in version 1.1, i.e., home network server, serving network server, and forwarding network server. However, the security among these three network servers is not specified in LoRaWAN specification. In this paper, a secure session keys establish method, named network Server Session Keys Establish (SSKE) method, is proposed to generate multiple session keys for three different types of network servers so that they can communicate with each other by using these session keys. With the assistance of join server, the key establish process employs the elliptic curve cryptography, two-dimensional operations, and time keys, to exchange their session keys. The SSKE not only effectively hides important encryption parameters, but also achieves fully mutual authentication among three servers. Security analysis shows that the SSKE can resist known-key, impersonation, replay, eavesdropping, and forgery attacks. Moreover, the SSKE generates 40 session keys in a key establish process, meaning the proposed protocol can support 40 sessions simultaneously.

Keywords: LoRaWAN · Security · Network server · Join server · Session key

1 Introduction

Nowadays, various Internet of Things (IoT) applications enhance human beings' quality of lives gradually. For example, IoT based smart city [1, 2] provides an intelligent scheme to manage transportation, citizens' healthcare, energy consumption, living environment, etc.; IoT factory [3] permits the products with improved quality and lower cost by leveraging the data collected by IoT. The development of IoT comes from the advancement of various technologies, including sensors, wireless communication technologies, security policies, innovative applications, and so on. Among them, wireless communication technologies play a very important role.

I. You et al. (Eds.): MobiSec 2019, CCIS 1121, pp. 23–33, 2020.
https://doi.org/10.1007/978-981-15-9609-4_3

Low-Power Wide-Area Network (LPWAN) is a wireless telecommunication wide area network designed to allow long range communications at a low bit rate among connected objects. Some LPWAN specifications, such as Narrow Band IoT (NB-IoT) [4], LoRaWAN [5], Sigfox [6], Telensa [7], and Weightless [8], have been proposed for IoT data communication. Among them, LoRaWAN, using unlicensed bands to define IoT network architecture and communication scheme, has many attractive features, such as long-range communication, long battery lifetime, secure data communication and high network capacity.

A typical LoRaWAN topology includes numerous end-devices, several gateways, network servers, application servers, and a join server. According to the specifications of the LoRaWAN [9, 10], the LoRaWAN utilizes Advanced Encryption Standard (AES) [11] to secure payload of a message transmitted between end-devices and application servers and to guarantee message integrity between end-devices and network servers. However, as mentioned by LoRa Alliance, the secure communication method between network server and join server is not specified in the LoRaWAN specification [9].

In this paper, a secure key generation and renew method, named Secure Communication for LoRaWAN Servers (SeCo for short), is proposed to provide a secure AES encryption/decryption key generation procedure and key renew procedure between LoRaWAN's network server and join server. The SeCo uses a key renew counter, time keys, random numbers, and binary operations to prevent the procedure suffering replay and eavesdropping attacks. Besides, AES is also utilized in the SeCo to encrypt important information, and no other complex encryption/decryption cryptography is needed. Security analysis shows that the SeCo can achieve mutual authentication, provide message integrity, and resist replay and eavesdropping attacks.

The rest of the paper is organized as follows. Section 2 briefly introduces the LoRaWAN architecture and its security scheme. Besides, some related studies are also investigated in Sect. 2. Section 3 presents the SeCo and its security is discussed in Sect. 4. Finally, Sect. 5 concludes this paper and describes some future studies.

2 Preliminary

In this section, we first introduce the LoRaWAN architecture and its security scheme, and then discuss some related studies of this work.

2.1 LoRaWAN Architecture and Its Security Scheme

LoRaWAN, developed by LoRa Alliance, is an attractive LPWAN protocol. Generally, there are numerous end-devices, several gateways, network servers, application servers, and a join server in a LoRaWAN environment. The end-devices communicated with gateways can be sensors, meters, monitors, controllers, machines, and so on. Gateways pass messages sent by end-devices to the network server and then the network server verifies messages' integrity and delivers these messages to corresponding application servers. Application server responses with the corresponding action based on the information carried in the receiving messages. Join server manages the end-devices join process and

generates two session keys, NwkSKey and AppSKey, for network server and application server, respectively.

The LoRaWAN security policy uses standardized AES cryptographic algorithm and end-to-end secure communication protocols to achieve the requirements of mutual authentication, confidentiality and integrity protection. Two operations, i.e., Cipher-based Message Authentication Code (CMAC) and Counter Mode (CTR), are combined with original AES encryption/decryption algorithm so as to perform message integrity protection and data encryption. During new end-device joining process, two unique 128-bit root keys, AppKey and NwkKey (both equipped with new end-device and recorded in join server), and a globally unique identifier EUI-64-based DevEUI (also equipped with new end-device) are utilized to generate several session keys. They are

- Network Session Key(s) (NwkSKey for LoRaWAN 1.0 and SNwkSIntKey, FNwkSIntKey, NwkSEncKey for LoRaWAN 1.1) which is(are) a(three) unique 128-bit key(s) shared by the end-devices and network server(s), and
- Application Session Key (AppSKey) which is a unique 128-bit key shared by end-device and the corresponding application server.

Figure 1 shows the traffic between end-device and application server is protected using these session keys. Each payload is encrypted by AES-CTR and carries a frame counter (to avoid packet replay) and a Message Integrity Code (MIC) computed with AES-CMAC (to avoid packet tampering). While the payload or MAC Header/Frame Header is tampered, the receiver cannot compute the correct MIC so as to guarantee communication data integrity. As mentioned above, AES algorithms are used to provide authentication and integrity of packets to the network server and end-to-end encryption to the application server. Although the LoRaWAN specifies the communication security between end-devices and application servers and between end-devices and network servers, the security between application server and join server is not defined in its specifications [9, 10].

2.2 Related Studies

IoT demonstrates a great convenience to many people's lives, however, due to the heterogeneous nature and constrained resource of IoT devices, the security and privacy problems threaten an IoT-based system [12, 13]. For example, Kotenko *et al.* [14] described the attack problem on IoT network layer and specified 14 types of attack behaviours. Hui *et al.* [15] also summarized many IoT related security challenges, e.g., key management, intrusion detection, access control, privacy protection. They also discussed the technical characteristics of blockchain, and described how to utilize these characteristics in IoT.

Although the LoRaWAN utilizes AES and SSL to secure IoT data communication, many studies [16–18] pointed out several weaknesses of LoRaWAN security. Butun *et al.* [16] analyzed security risks of LoRaWAN v1.1 by using ETSI guidelines and created a threat catalog for this system where the security risks, coming from new security framework and incomplete LoRaWAN specification, comprise vulnerabilities against end-device physical capture, rogue gateway and replay attack. Miller [17] introduced some possible attacks on LoRaWAN, and recommended that the session key management

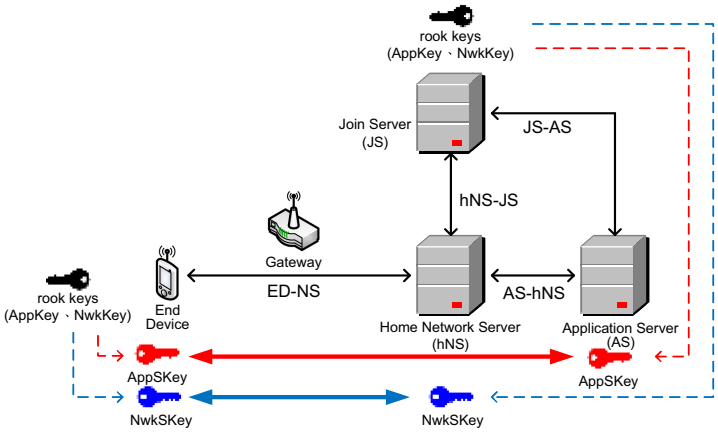

Fig. 1. Two session keys are used for end-to-end data security.

policy as well as its session key generation process should be improved. You *et al.* [18] examined the standard LoRaWAN protocol, and observed that it fails to support the perfect forward secrecy, the end-to-end security and the defense against the replay attack.

In order to enhance the security of LoRaWAN, many studies [18–20] proposed new methods for data encryption, session key management, and session key update/generation. [18] utilized default option and security-enhanced option to prevent a malicious network server from breaking the end-to-end security. Sanchez-Iborra *et al.* [19] evaluated the security vulnerabilities of LoRaWAN in the area of key management and proposed a lightweight key management method on Ephemeral Diffie-Hellman Over COSE transaction. To enhance the security of AES key generation process, Hayati *et al.* [20] investigated several parameters, e.g., key generation time, randomness level, and key length, and claimed that these parameters should be considered in the key generation process.

In spite of previous studies provided higher security level for LoRaWAN environment, most of them considered end-to-end security, i.e., end-devices and application servers. Only little attention had been given to the point of server to server security. As a result, we propose the SeCo to provide secure communication method for LoRaWAN's servers.

3 Secure Communication Method for LoRaWAN Servers

In order to create a secure communication channel for LoRaWAN's join server and network server, a special data encryption/decryption key, i.e., *NJKey*, is generated at first time and then renewed periodically. Once, the key is generated, the important information and commands between join server and network server can be protected by using this key. The key generation procedure and key renew procedure are introduced in this section.

3.1 Key Generation Procedure

Key generation procedure is used to generate a communication key between join server and network server when the LoRaWAN is built at the first time. In the key generation procedure, a per-installed key, *NSJSKey*, and a random number, r_A, are utilized to produce a new encryption/decryption key, i.e., *NJKey*. As shown in Fig. 2, there are four rounds in the key generation procedure.

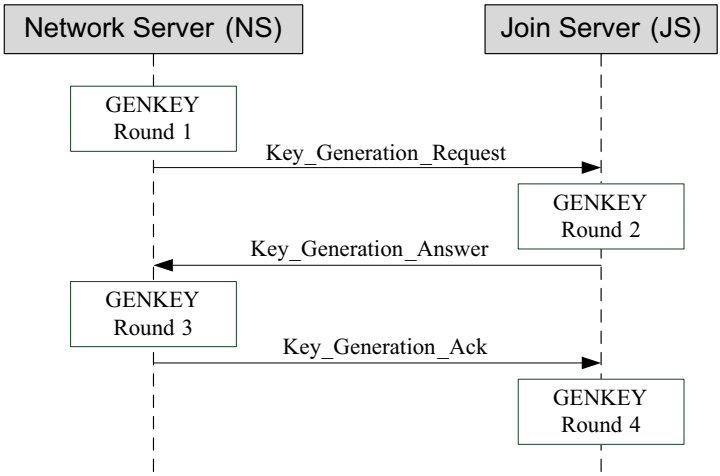

Fig. 2. The key generation procedure of the SeCo.

– GENKEY Round 1:

The Network Server (NS)

1. fetches the system time $t_{nonce,NS}$ and generates the time key K_T;
2. generates a random number $r_A \in Z_n^*$;
3. generates a key renew counter *NSJSCounter* and initializes its value to be 1;
4. calculates $Address\_checking = aes128\_encrypt(K_T, NSAddr||JSAddr||NSJSCounter)$, where *NSAddr*, *JSAddr* are the addresses of network server and join server, and $aes128\_encrypt$ represents message encryption by using 128-bit AES cryptography algorithm with encryption key K_T;
5. calculates $GenKeyMsg = (NSAddr||JSAddr||(r_A +_2 (NSJSKey \oplus NSJSCounter))) \oplus K_T$, where *NSJSKey* is a pre-installed key for both network server and join server, $+_2$ and $\oplus$ indicate binary addition and binary exclusive-or operation, respectively;
6. calculates $GenKey\_req = aes128\_encrypt(NSJSKey \oplus K_T, GenKeyMsg)$;
7. sends Key_Generation_Request $= \{t_{nonce,NS}, Address\_checking, GenKey\_req\}$ to the join server JS.

– GENKEY Round 2:

When receiving the messages, the Join Server JS

1. fetches the system time $t_{nonce,JS}$ and undergoes a time condition by checking to see whether or not $t_{nonce,JS}$ satisfies $t_{nonce,JS} - t_{nonce,NS} \leq \delta_t$, where δ_t is a predefined time threshold for the allowable maximum transmission delay from NS to JS; If checking failed, it discards this message and sends an GenKeyFail message to NS. Otherwise, goes to next step;
2. derives K_T from $t_{nonce,NS}$;
3. obtains *NSAddr*, *JSAddr*, and *NSJSCounter* by calculating $NSAddr||JSAddr||NSJSCounter = aes128\_decrypt(K_T, Address\_checking)$, where *aes*128*_decrypt* represents message decryption by using 128-bit AES cryptography algorithm with decryption key K_T, and then checks to see whether *NSAddr* and *JSAddr* are recorded in its database or not; If not, it discards this message and sends an GenKeyFail message to NS. Otherwise, goes to next step;
4. fetches *NSJSKey* from its database and calculates $GenKeyMsg = aes128\_decrypt(NSJSKey \oplus K_T, GenKey\_req)$;
5. obtains *NSAddr*, *JSAddr*, from *GenKeyMsg* (step (4)) and check to see whether or not these two addresses are the same with step (3); If not, it discards this message and sends a GenKeyFail message to NS. Otherwise, goes to next step;
6. calculates $r_A = ((GenKeyMsg \oplus K_T) - NSAddr||JSAddr) -_2 (NSJSKey \oplus NSJSCounter)$;
7. fetches another system time $t'_{nonce,JS}$;
8. calculates $GenKey\_Ans = aes128\_encrypt(NSJSKey \oplus r_A, NSJSCounter||t'_{nonce,JS})$;
9. sends Key_Generation_Answer $= \left\{t'_{nonce,JS}, GenKey\_Ans\right\}$ to NS.

– GENKEY Round 3:

When receiving the messages sent from JS, the Network Server NS

1. fetches the system time $t'_{nonce,NS}$ and undergoes a time condition by checking to see whether or not $t'_{nonce,JS}$ satisfies $t'_{nonce,NS} - t'_{nonce,JS} \leq \delta_{t'}$, where $\delta_{t'}$ is a predefined time threshold for the allowable maximum transmission delay from JS to NS; If checking failed, it discards this message and sends an GenKeyFail message to JS. Otherwise, goes to next step;
2. calculates $NSJSCounter\_receive||t'_{nonce,JS}\_receive = aes128\_decrypt(NSJSKey \oplus r_A, GenKey\_Ans)$, where *NSJSCounter_receive* and $t'_{nonce,JS}\_receive$ mean *NSJSCounter* and $t'_{nonce,JS}$ receiving from the message sent by JS;
3. checks to see whether $NSJSCounter\_receive = NSJSCounter$ and $t'_{nonce,JS}\_receive = t'_{nonce,JS}$ or not; If not, it discards this message and sends an GenKeyFail message to JS. Otherwise, goes to next step;
4. generates the AES data encryption/decryption key $NJKey = (NSJSKey +_2 r_A) \oplus r_A$;

5. calculates $GenKey\_Ack = aes128\_encrypt\left(NJKey, t'_{nonce,JS}\right)$;
6. sends Key_Generation_Ack = $\{GenKey\_Ack\}$ to JS.

– GENKEY Round 4:

When receiving the message, the Join Server JS

1. generates the data encryption/decryption key $NJKey = (NSJSKey +_2 r_A) \oplus r_A$;
2. calculates $t'_{nonce,JS}\_receive = aes128\_decrypt(NJKey, GenKey\_Ack)$;
3. checks to see whether $t'_{nonce,JS}\_receive = t'_{nonce,JS}$ (in step (7) of Round 2) or not; If not, it sends a restart message to NS. Otherwise, stores $NJKey$ as data encryption/decryption key.

3.2 Key Renew Procedure

In order to enhance communication security, the AES encryption/decryption key $NJKey$ is renewed periodically. Figure 3 shows the key renew procedure in which it has three rounds, and each round has several steps.

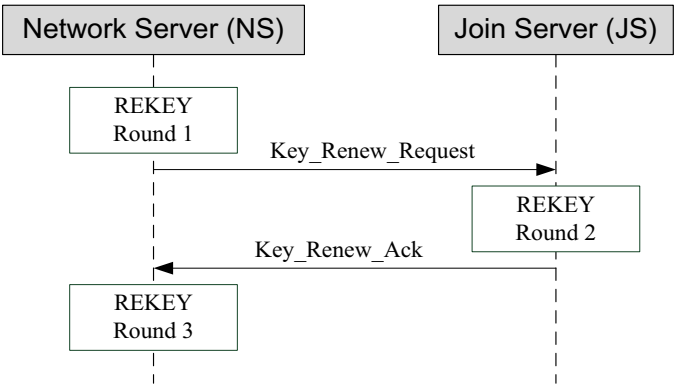

Fig. 3. The SeCo key renew procedure.

– REKEY Round 1:

When key renew time is up or JS sends a key renew message, the network server NS

1. fetches the system time $t''_{nonce,NS}$ and generates the time key $K_{T''}$;
2. generates a random number $r_B \in Z_n^*$;
3. fetches $NSJSCounter$ and r_A from its database and calculates $NSJSCounter\_new = NSJSCounter +_2 1 +_2 r_A$;
4. calculates $ReKey\_req = aes128\_encrypt(NJKey, NSJSCounter\_new||r_B)$;

5. sends Key_Renew_Request $= \left\{ t''_{nonce,NS}, ReKey\_req \right\}$ to the join server JS.

– REKEY Round 2:

When receive the messages send from NS, the Join Server JS

1. fetches the system time $t''_{nonce,JS}$ and undergoes a time condition by checking to see whether or not $t''_{nonce,JS}$ satisfies $t''_{nonce,JS} - t''_{nonce,NS} \leq \delta_t$; If checking failed, it discards this message and sends an ReKeyFail message to NS. Otherwise, goes to next step;
2. calculates $NSJSCounter\_new\_receive || r_B = aes128\_decrypt(NJKey, ReKey\_req)$;
3. fetches $NSJSCounter$ and r_A from its database and calculates $NSJSCounter\_new = NSJSCounter +_2 1 +_2 r_A$; and then checks to see whether $NSJSCounter\_new\_receive = NSJSCounter\_new$ or not; If not, it discards this message and sends an ReKeyFail message to NS. Otherwise, it updates $NSJSCounter$ as $NSJSCounter +_2 1$ and goes to next step;
4. generates new encryption/decryption key $NJKey\_new = (NJKey +_2 r_B) \oplus r_A$;
5. calculates $ReKey\_ack = aes128\_encrypt(NJKey\_new, NSAddr \oplus r_B)$;
6. stores $NJKey\_new$ as new $NJKey$, replaces r_A with r_B, updates $NSJSCounter$ as $NSJSCounter +_2 1$ in its database and sends Key_Renew_Ack $= \{ReKey\_ack\}$ to NS.

– REKEY Round 3:

Once receiving the messages send from JS, the network server NS

1. generates the new encryption/decryption key $NJKey\_new = (NJKey +_2 r_B) \oplus r_A$;
2. calculates $NSAddr = (aes128\_decrypt(NJKey\_new, ReKey\_ack)) \oplus r_B$;
3. checks to see whether $NSAddr$ is correct or not; If not, it discards this message and sends an ReKeyFail message to JS to restart key renew procedure. Otherwise, it stores $NJKey\_new$ as new $NJKey$, replaces r_A with r_B, and updates $NSJSCounter$ as $NSJSCounter +_2 1$ in its database.

4 Security Analysis

This section analyzes the security features of the SeCo, including mutual authentication, confidentiality and integrity protection, replay attack resistance and eavesdropping attack resistance.

- Mutual authentication

Mutual authentication which established between network server and join server ensures that only genuine and authorized servers can perform the key generation procedure and key renew procedure. Firstly, a network server and join server pair equip with a pre-installed key, i.e. $NSJSKey$, which is utilized to encrypt messages in the key generation

procedure. Only an authenticated server can decrypt the messages with correct *NSJSKey* and pass the checking in step 5 of GENKEY Round 2, step 3 of GENKEY Round 3, and step 3 of GENKEY Round 4. Secondly, after key generation procedure, a data encryption key *NJKey* is generated by using *NSJSKey* and the random number r_A. When this data encryption key needs to be renewed, in step 3 of REKEY Round 2, the join server verifies *NSJSCounter* and previous r_A, which are both stored in join server's database, to authorize network server. In step 3 of REKEY Round 3, the network server authorizes join server by checking *NSAddr* which is encrypted with *NJKey_new*, and the *NJKey_new* is generated by previous *NJKey*, r_B and r_A. Only the authorized network server and join server have these parameters and can decrypt the correct *NSAddr*. In summary, the network server and join server authenticate with each other by using *NSJSKey*, *NSJSCounter*, r_A, r_B, and *NSAddr*.

- Confidentiality

In the SeCo, all of the important messages are encrypted by using 128-bit AES cryptography algorithm except system time t_{nonce} which is utilized for resisting replay attack. Moreover, the AES encryption key in each round is different; K_T is used in step 4 of GENKEY Round 1; $NSJSKey \oplus K_T$ is utilized in step 6 of GENKEY Round 1; $NSJSKey \oplus r_A$ is employed in step 8 of GENKEY Round 2; *NJKey* is applied in step 5 of GENKEY Round 3 and step 4 of REKEY Round 1; and *NJKey_new* is operated in step 5 of REKEY Round 2. Since 128-bit AES is a well-know and high secure level cryptography algorithm, the parameters and information can be encrypted during key generation procedure and key renew procedure. Besides, in the SeCo, the encryption/decryption key is renewed periodically by using key renew procedure so as to enhance the communication security and provide high confidentiality for LoRaWAN.

- Integrity protection

Message integrity protection indicates that a message has not been tampered with or altered during transmission. The most common approach is to use a hash function that combines all the bytes in the message with a secret key and produces a message digest that is difficult to reverse. In order to simplify the key generation procedure and key renew procedure, in the SeCo, the *NSAddr* and *JSAddr* are used to guarantee the message integrity in step 3 and 5 of GENKEY Round 2, and the $t'_{nonce,JS}$ is also utilized for message integrity protection in step 3 of GENKEY Round 3 and step 4 of GENKEY Round 4. The key renew procedure also employs *NSJSCounter* and r_A in step 3 of REKEY Round 2, and adopts *NSAddr* in step 3 of REKEY Round 3 to protect message integrity. It follows from what has been said that all of the message receivers in the SeCo verify messages' integrity when they receiving the message.

- Replay attack resistance

In the key generation procedure, the time key K_T is derived from the network server's system time $t_{nonce,NS}$. A replay attack is that a hacker duplicates a valid message transmitted by the network server, and pretends the legal network server to send the message to join server so as to obtain related information. Two situations may occur. The first on is the hacker transmits the original message to the join server without modifying it. However, the checking in step 1 of GENKEY Round 2 $t_{nonce,JS} - t_{nonce,NS} \leq \delta_t$ cannot be held since the retransmission delay will make $t_{nonce,JS} - t_{nonce,NS} > \delta_t$. The second situation is the hacker modifies the time $t_{nonce,NS}$ to make the condition of $t_{nonce,JS} - t_{nonce,NS} \leq \delta_t$ hold. Nevertheless, in step 2 of GENKEY Round 2, the join server uses $t_{nonce,NS}$ to derive K_T which is then utilized to decrypt three parameters, i.e., *NSAddr*, *JSAddr*, and *NSJSCounter*. The decrypted network server address *NSAddr* and join server address *JSAddr* are compared with those parameters in join server's database. Once the K_T is a incorrect decryption key, the verification is failed. In step 1, 2 and 3 of GENKEY Round 3, the network server also resists replay attacks by using $t'_{nonce,JS}$. Similarly, in REKEY Round 1 and Round 2, the network server and join server adopt $t''_{nonce,NS}$ to prevent replay attacks.

Furthermore, in the SeCo, a lifetime counter, *NSJSCounter*, used for recording a unique number of key generation/renew procedure is utilized both in key generation procedure and key renew procedure. *NSJSCounter* is initially set to 1, and then increased by previous procedure's ransom number r_A in step 3 of REKEY Round 1. Since this counter is managed by network server and join server, and is encrypted within transmitted messages, when a hacker catches and duplicates a valid message, and then he re-transmits this message to join server or network server, the value of *NSJSCounter* is equal or less then the value in the message. It indicates the received message is not from a genuine and authentic server, thus this counter can also be used to resist replay attack.

- Eavesdropping attack resistance

A hacker may extract important information when he/she captures a large amount of messages from the underlying network. The most important information we need to protect is the message encryption/decryption key *NJKey*. In the SeCo, new *NJKey* is generated by using previous *NJKey* and two random numbers r_A and r_B. While r_A is generated in key generation procedure (or last key renew procedure), and r_B is generated in current key renew procedure, and both r_A and r_B are protected by using AES algorithm, the hacker is unable to extract one of these three parameters from the captured messages. Thus, the SeCo is invulnerable to the eavesdropping attack.

5 Conclusion and Future Studies

To provide secure communication channel between LoRaWAN's network server and join server, the SeCo is proposed in this study. A data encryption/decryption key is generated in the key generation procedure and updated periodically by using key renew procedure. Two random numbers and one key-renew counter are utilized to guarantee

the signal integrity during the key generation procedure and key renew procedure. The security analysis shows that the SeCo can provide mutual authentication, confidentiality and message integrity, and also can resist replay attack and eavesdropping attack.

In the future, we would like to simplify the key generation procedure so that the secure communication channel between network server and join server can be created quickly. Besides, the security issues among application server, join server, and network servers will also be investigated. These constitute our future studies.

References

1. Gaur, A., Scotney, B., Parr, G., McClean, S.: Smart city architecture and its applications based on IoT. Proc. Comput. Sci. **52**, 1089–1094 (2015)
2. Shih, C.-S., Chou, J.-J., Lin, K.-J.: WuKong: Secure Run-Time environment and data-driven IoT applications for Smart Cities and Smart Buildings. J. Internet Serv. Inf. Secur. **8**(2), 1–17 (2018)
3. Chekired, D.A., Khoukhi, L., Mouftah, H.T.: Industrial IoT data scheduling based on hierarchical fog computing: a key for enabling smart factory. IEEE Trans. Industr. Inf. **14**(10), 4590–4602 (2018)
4. Flore, D.: 3GPP Standards for the Internet-of-Things, Recuperado el 25 (2016)
5. Lora-alliance. https://www.lora-alliance.org. Accessed 15 Aug 2019
6. Sigfox. https://www.sigfox.com. Accessed 15 Aug 2019
7. Telensa. http://www.telensa.com. Accessed 15 Aug 2019
8. Weightless. http://www.weightless.org. Accessed 15 Aug 2019
9. LoRa Alliance Technical Committee: LoRaWAN Backend Interfaces 1.0 Specification. LoRa Alliance (2017)
10. LoRa Alliance Technical Committee: LoRaWAN 1.1 Specification. LoRa Alliance (2017)
11. Announcing the Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197. United States National Institute of Standards and Technology (2001)
12. Korzhuk, V., Groznykh, A., Menshikov, A., Strecker, M.: Identification of attacks against wireless sensor networks based on behaviour analysis graphics processing units. J. Wirel. Mob. Netw. Ubiquit. Comput. Depend. Appl. **10**(2), 1–21 (2019)
13. Gritti, C., Önen, M., Molva, R., Susilo, W., Plantard, T.: Device identification and personal data attestation in networks. J. Wirel. Mob. Netw. Ubiquit. Comput. Depend. Appl. **9**(4), 1–25 (2018)
14. Kotenko, I., Saenko, I., Branitskiy, A.: Applying big data processing and machine learning methods for mobile Internet of Things security monitoring. J. Internet Serv. Inf. Secur. **8**(3), 54–63 (2018)
15. Hui, H., et al.: Survey on blockchain for Internet of Things. J. Internet Serv. Inf. Secur. **9**(2), 1–30 (2019)
16. Butun, I., Pereira, N., Gidlund, M.: Security risk analysis of LoRaWAN and future directions. Fut. Internet **11**(1), 1–22 (2019). Article ID 3
17. Miller R.: LoRa Security – Building a Secure LoRa Solution. MWR Labs, Whitepaper (2016)
18. You, I., Kwon, S., Choudhary, G., Sharma, V., Seo, J.: An enhanced LoRaWAN security protocol for privacy preservation in IoT with a case study on a smart factory-enabled parking system. Sensors **18**(6), 1–32 (2018). Article ID 1888
19. Sanchez-Iborra, R., et al.: Enhancing LoRaWAN security through a lightweight and authenticated key management approach. Sensors **18**(6), 1–18 (2018). Article ID 1833
20. Hayati, N., Suryanegara, M., Ramli, K., Suryanto, Y.: Potential development of AES-128-bit key generation for LoRaWAN security. In: International Conference on Communication Engineering and Technology Proceedings, Nagoya, Japan, pp. 57–61 (2019)