Computers, Materials & Continua
DOI: 10.32604/cmc.2022.026557
Article

Tech Science Press

Two-Stage High-Efficiency Encryption Key Update Scheme for LoRaWAN Based IoT Environment

Kun-Lin Tsai[1,2,*], Li-Woei Chen[3], Fang-Yie Leu[4,5] and Chuan-Tian Wu[1]

[1]Department of Electrical Engineering, Tunghai University, Taichung, 407, Taiwan
[2]Research Center for Smart Sustainable Circular Economy, Tunghai University, Taichung, 407, Taiwan
[3]Department of Computer and Information Sciences, Chinese Military Academy, Kaohsiung, 830, Taiwan
[4]Department of Computer Science, Tunghai University, Taichung, 407, Taiwan
[5]Emergency Response Management Center, Industry-Academia Collaboration and University Extension Division, Ming-Chuan University, Taipei, 111, Taiwan
*Corresponding Author: Kun-Lin Tsai. Email: kltsai@thu.edu.tw
Received: 30 December 2021; Accepted: 02 March 2022

Abstract: Secure data communication is an essential requirement for an Internet of Things (IoT) system. Especially in Industrial Internet of Things (IIoT) and Internet of Medical Things (IoMT) systems, when important data are hacked, it may induce property loss or life hazard. Even though many IoT-related communication protocols are equipped with secure policies, they still have some security weaknesses in their IoT systems. LoRaWAN is one of the low power wide-area network protocols, and it adopts Advanced Encryption Standard (AES) to provide message integrity and confidentiality. However, LoRaWAN's encryption key update scheme can be further improved. In this paper, a **T**wo-stage **H**igh-efficiency LoRaWAN encryption key **U**pdate **S**cheme (THUS for short) is proposed to update LoRaWAN's root keys and session keys in a secure and efficient way. The THUS consists of two stages, i.e., the Root Key Update (RKU) stage and the Session Key Update (SKU) stage, and with different update frequencies, the RKU and SKU provide higher security level than the normal LoRaWAN specification does. A modified AES encryption/decryption process is also utilized in the THUS for enhancing the security of the THUS. The security analyses demonstrate that the THUS not only protects important parameter during key update stages, but also satisfies confidentiality, integrity, and mutual authentication. Moreover, The THUS can further resist replay and eavesdropping attacks.

Keywords: Key update; AES; LoRaWAN; IoT security; high efficiency

1 Introduction

Due to the rapid developments of Internet of Things (IoT), Artificial Intelligence (AI), and communication technologies, in recent years, IoT applications have been built for different fields. For example, an IoT system can be used in industry and medical treatment, and forms the so-called

This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Industrial Internet of Things (IIoT) [1] and Internet of Medical Things (IoMT) [2] systems. For human daily lives, a smart home system [3] and a smart city [4] can also be created by combining IoT and AI technologies. In an IoT system, massive sensors are linked to network via wired and wireless connections, and then sensed data are delivered to control center, so that the corresponding operations can be performed.

In order to provide mass, wide-range and low-power connections, the concept of Low Power Wide Area Network (LPWAN) communication structure is established. In the past decade, many LPWAN communication systems have been proposed, e.g., Narrow Band IoT (NB-IoT) [5], Long Range Wide-Area Network (LoRaWAN) [6], Sigfox [7], DASH7 [8], etc. Due to business and research reason, many NB-IoT and LoRaWAN related studies have been proposed. To allow mass IoT devices to communicate with each other, NB-IoT utilizes licensed telecommunication bands, while LoRaWAN uses unlicensed band. Generally, both NB-IoT and LoRaWAN have the features of wide-area and low-energy communication. Moreover, LoRaWAN is an evolving protocol and can be used for many modern IoT environments.

The IoT security is as important as general computer and network security since its applications relate to businesses, industries, cities, and even human healthcare [9,10]. Providing information confidentiality, data integrity, and device availability becomes an essential requirement for LPWAN communication systems. The Advanced Encryption Standard (AES) [11] is adopted by LoRaWAN to ensure the transmitted data can be safely delivered from sender to receiver. Besides, two types of session keys are utilized in AES data encryption and decryption processes between end-devices and servers [12].

Even though LoRaWAN uses AES to protect the security of transmitted data, it still exists some security problems that need to be solved [13–15]. For example, LoRaWAN's root keys are kept in join-server and end-devices, but lack their own update mechanisms. Once the root keys are stolen, LoRaWAN's security might be crashed. Besides, the session keys are updated through the end-device re-join process which may also lead to device management and data consistency problems.

In order to deal with the key management and update problem, in this study, a **T**wo-stage **H**igh-efficiency LoRaWAN key **U**pdate **S**cheme (THUS for short) is proposed to update LoRaWAN's root keys and session keys in a secure and efficient way. The THUS utilizes modified AES process and system time stamp to achieve the goal of mutual authentication, confidentiality and message integrity during its update procedure. Besides, the security analysis also shows that the THUS has the ability to resist replay and eavesdropping attacks. In the THUS, the root key and session key can be renewed without changing the device information recorded in join-server, network-server, and application-server. Besides, with different key update frequencies, the THUS provides higher security level than the conventional LoRaWAN specification does. The main contributions of this paper are listed below.

(i) The proposed THUS improves LoRaWAN key management by updating both root keys and session keys.
(ii) The modified AES algorithm is utilized in the THUS so that the security of key update procedure can be enhanced and the processing performance can also be improved.
(iii) During the key update procedure, the proposed THUS provides the features of mutual authentication, confidentiality and message integrity and has the ability to resist eavesdropping attack and replay attack.

The remaining part of this paper is organized as follows. LoRaWAN security policy and some IoT-related studies are introduced in Section 2. Then, Section 3 presents the detailed process of the THUS,

including the Root Key Update (RKU) stage and Session Key Update (SKU) stage. The security analyses and discussion are presented in Section 4. Section 5 addresses the THUS performance. Section 6 concludes this study and points out our future study.

2 Preliminary

2.1 LoRaWAN Security

Since LoRaWAN is one of the communication protocols designed for IoT applications, it provides many attractive features including low power and long-distance communication. The security policy of LoRaWAN [16,17] is established under the requirements of data confidentiality, mutual authentication between devices and servers, message integrity. Accordingly, AES cryptographic algorithm and two operation modes, i.e., Counter Mode (CTR) and Cipher-based Message Authentication Code (CMAC), are adopted in the LoRaWAN data transmission. The CTR is used for data encryption and decryption, while the CMAC is utilized to guarantee the message integrity. Besides, LoRaWAN uses two types of session keys to protect data security, one for end-devices and application-servers, namely application session key (AppSKey), and another for end-devices and network-servers, namely network session key (NwkSKey for R1.0, and FNwkSIntKey, SNwkSIntKey, NwKSEncKey for R1.1. To simplify, in this study, the NwkSKey is used to represent all network session keys). As shown in Fig. 1, when an end-device new joins to an IoT system, a globally unique identifier DevEUI and two unique root keys, including AppKey and NwkKey, are registered in join-server. And then, the join process between end-device and join-server generates two types of session keys by using DevEUI, AppKey and NwkKey. Following that, the NwkSKey and AppSKey are delivered to network-server and application-server, respectively.

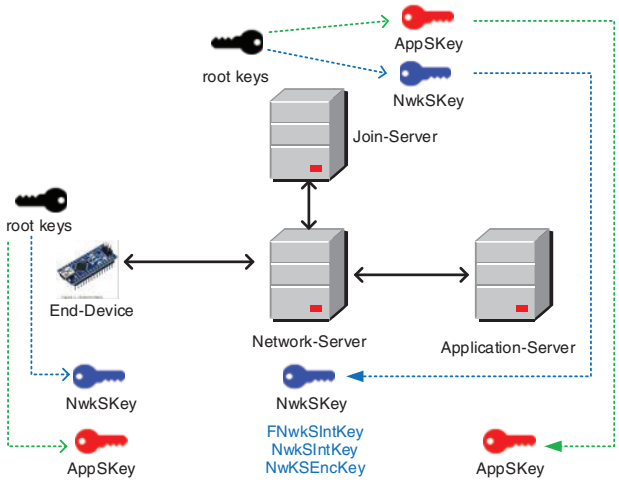

Figure 1: LoRaWAN uses two root keys to generate session keys

2.2 Related Studies

Since IoT security is an important topic for constructing an IoT system, many related studies [18–21] have been proposed in the past years. According to the announced LoRaWAN 1.0 protocol, Naoui et al. [18] pointed out two important parameters might be attacked. The first parameter is a 16-bit counter, *DevNonce*, which is used to record the join times of an end-device. Since the *DevNonce* is transmitted with unencrypted format, the replay attack might be launched by some attacker. The

second parameter is *AppNonce* which is utilized for mutual authentication between application-server and end-device. An attacker may forge the join acceptance message sent by network server, retransmit the same message to the end-device, and then pretend to be a legitimate application-server. In [18], Naoui et al. utilized an independent computer to be the trusted third party and created a timeline for every message so that the session keys can be delivered to network-server and application-server in a secure way.

Eldefrawy et al. [19] checked the security policies of different LoRaWAN protocols. Similarly, they also claimed that the transmitted data will be disclosed when the parameter *AppNonce* in LoRaWAN v1.0 is known by attackers. However, LoRaWAN v1.1 utilized join-server to generate and dispatch session keys, hence enhanced the security level. Even though LoRaWAN improves its security in v1.1, many researchers still think the root key management is a drawback of LoRaWAN's security policy.

Chen et al. [20] proposed a comprehensive LoRaWAN key management scheme, which addressed key updating, key generation, key backup, and key backward compatibility. The centralized key management server designed in [20] was a trusted agent dealing with the whole lifecycle of the key management, i.e., key generation/derivation, updating, backup/recovery, and key revocation. Compared with other previous studies, they provided a good and secure solution for LoRaWAN key management. However, involving an extra server may also lead to other security problems. Xing et al. [21] used elliptic curve cryptography for root key update scheme. Both end-devices and server adopted the hierarchical deterministic wallet to manage the key pair, and the key agreement between the end-devices and the server was realized by the elliptic curve Diffie-Hellman key exchange algorithm. They also used bi-directional Hash algorithm during communication. Although their method indeed enhanced the security of LoRaWAN key management, the complex elliptic curve cryptography algorithm consumed much power during key update procedure.

Dönmez et al. [22] utilized a master device to store the original root keys of end-devices so as to enhance the LoRaWAN's security. The connections between master device and end-devices are physical links, as a result, the master device can be considered as an extension part of end-devices. Moreover, the master device can also be utilized to charge the batteries of end-devices. However, when master device was hacked, it may result in serious problem of whole IoT system.

Because original root keys are stored in end-devices, and in LoRaWAN standard, only session key can be update by using rejoin process; however, the root keys cannot be update. In [23], Han and Wang created a root key update method by using key derivation function to solve the key management problem. Their experiment showed that the key generation process can generate new root keys with very high degree of randomness.

3 Two-Stage LoRaWAN Key Update Scheme

As abovementioned, LoRaWAN's protocols [16,17] do not take root key management into account. Once the root keys are attacked and stolen, it can easily induce security problems. In this section, as shown in Fig. 2, a two-stage LoRaWAN key update scheme is proposed. The first one is root key update (RKU) stage, and the second one is session key update (SKU) stage. In Section 3.1, the modified AES is introduced so as to enhance the security level and reduce the computational complexity at the same time. Then, the detailed steps of RKU stage and SKU stage are described in Sections 3.2 and 3.3, respectively. The notations used in the RKU and SKU stage are illustrated and defined in Tab. 1.

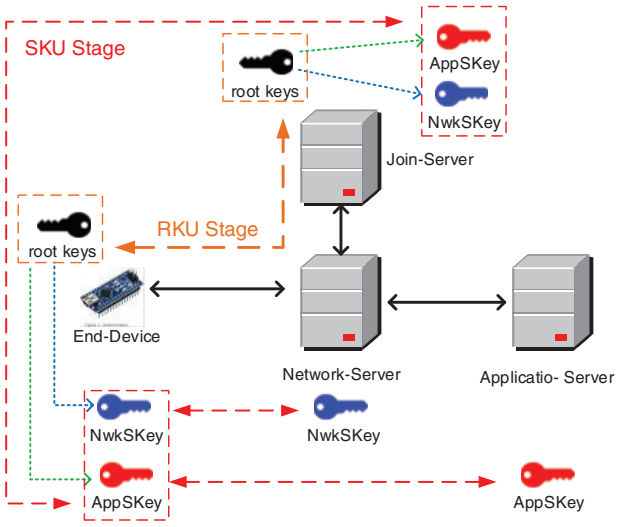

Figure 2: Root keys are updated in RKU stage and session keys are updated in SKU stage

Table 1: Notations used in RKU stage and SKU stage

Notations	Descriptions
$\alpha, \beta, \gamma, \epsilon$	128-bit random numbers
$t^{RKU}_{nonce,ns}$, $t^{RKU}_{nonce,js}$, $t^{RKU}_{nonce,ed}$	System time of network-server, join-server, and end-device in RKU procedure, respectively.
DevNonce	A 16-bit counter used to calculate the end-device's join time
R−Box	A randomly generated box used to calculate dynamic box D-box
NwkSKey	The network session key which is generated by the root key and is used to encrypt data delivered between end-device and network-server
DevEUI	A global end-device ID in IEEE EUI64 address space that uniquely identifies an end-device
k_{ns-js}	symmetric data encryption key for network-server and join-server
k^{ns}_{ct}, S^{js}_{ct}	Time keys of network-server and join-server
$En-D_{ed}$, $En-D_{js}$	Encrypted data for end-device and join-server
$D-Box_{new}$	New substitution box for AES SubBytes step
Δt_1, Δt_2, Δt_3, Δt_4, Δt_5, Δt_6, Δt_7, Δt_8	Predefined delay time
AppSKey	The application session key which is generated by the root key and is used to encrypt data transmitted between end-device and application-server
$En-Pr_{ed}$, $En-Pr_{js}$	Encrypted data generated by end-device and join-server
AppKey, ***NwkKey***	LoRaWAN root keys
JoinEUI	A global application ID in IEEE EUI64 address space that uniquely identifies the join-server

(Continued)

Table 1: Continued

Notations	Descriptions
$t^{SKU}_{nonce,js}$, $t^{SKU}_{nonce,ed}$, $t^{SKU}_{nonce,js}$	System time of join-server, end-device, and join-server in SKU procedure, respectively.
Pad_{16}	Appended zero to make the length of transmitted message as a multiple of 16
$En{-}K_{js2ed}$, $En{-}K_{js2as}$, $En{-}K_{js2ns}$	Encrypted data sent by join-server to end-device, application-server, and network-server, respectively.
$En{-}K_{as2ed}$, $En{-}K_{ns2ed}$	Encrypted data sent by application-server and network-server to end-device
Ack_{ed2js}	Acknowledge message

3.1 Modified AES

In order to improve AES's security strength and operating speed, a dynamic-box (D-Box) is designed to replace the original substitution-box (S-Box). The enhanced dynamic accumulated shifting substitution (EDASS) algorithm proposed in [24] and used in [25] is a randomized function which can be used to generate irreversible text. The EDASS based D-Box Generation (DBG) process uses three inner keys and three insertion arrays to generate a D-Box. During the RKU and SKU stage, the inputs of modified AES are plaintext and a D-Box, and the corresponding output is a ciphertext. Besides, a flag array is also utilized to prevent duplicate elements appear in the D-Box. For security reason, the D-Box is required irregular renewal. In this study, the D-Box of modified AES has 256 elements, which means that a total number of 256! ($=\prod_{i=1}^{256} i$) possible combinations of a D-Box. When an attacker captures some encrypted messages and would like to decrypt them, he needs 128-bit session key and the D-Box, indicating that the combination is up to $2^{128} \times 256!$. Since EDASS algorithm and DBG process are input-sensitivity and randomness, decrypting the encrypting messages in a short period of time without correct session key and D-Box are very difficult. Besides, as shown in [25], by replacing S-Box with the D-Box, the encryption cycles of the conventional 128-bit AES can be simplified to 5 rounds. As a result, the modified AES with D-Box has high efficiency than the conventional AES does.

3.2 Stage 1: Root Key Update Procedure

In the RKU stage, there are three important parameters, i.e., α, β, and γ, which are generated by network-server, end-device, and join-server, respectively. Firstly, network-server delivers α to join-server and end-device so as to state up the RKU procedure. Then, the end-device sends β to join-server by integrating α into the encrypted message so that the feature of message integrity can be achieved. At the same time, the join-server also transmits γ to end-device. Once when end-device and join-server have β and γ in hand, they utilize these two parameters to generate two new root keys. As shown in Fig. 3, in the RKU stage, there are 8 steps, which are described as follows.

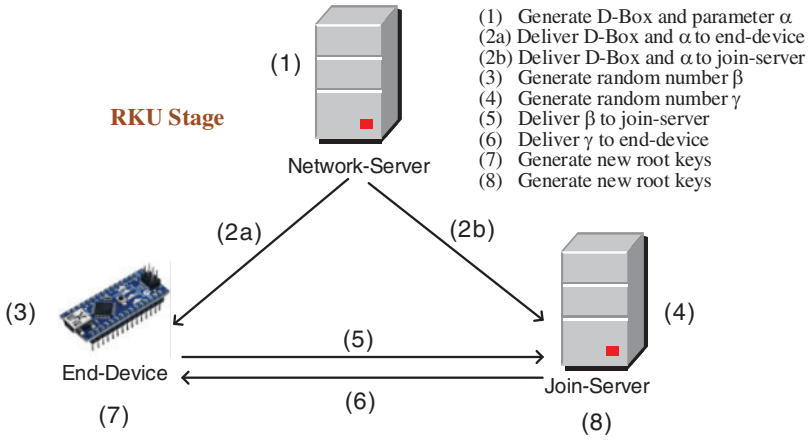

Figure 3: There are 8 steps in the RKU stage

Step (1): The network-server performs the following sub-steps.

(1.1) generates a random box $R-Box$ and a 128-bit random parameter α;
(1.2) fetches the system time $t^{RKU}_{nonce,ns}$;
(1.3) calculates the time key $k^{ns}_{ct} = SHA\left(t^{RKU}_{nonce,ns} \oplus \alpha \oplus DevNonce\right)$, where SHA stands for secure hash algorithm, and $DevNonce$ is a 16-bit counter;
(1.4) calculates new AES substitution box $D-Box_{new} = BDG\left(k^{ns}_{ct},\ R-Box\right)$;
(1.5) calculates $En-D_{ed} = EDASS\left(NwkSKey_{old},\ D-Box_{old}, k^{ns}_{ct} |D-Box_{new}| t^{RKU}_{nonce,ns} |\alpha|\ \text{DevEUI}\ \right)$, where $NwkSKey_{old}$ is currently used network session key and $DevEUI$ represents the global end-device ID in IEEE EUI64 address space that uniquely identifies the end-device;
(1.6) calculates $En-D_{js} = EDASS\left(k_{ns-js},\ D-Box_{old}, k^{ns}_{ct} |D-Box_{new}| t^{RKU}_{nonce,ns} |\alpha|\ DevEUI\right)$, where k_{ns-js} indicates the symmetric data encryption key of network-server and join-server.

Step (2a): The network-server delivers $En-D_{ed}$ to end-device.

Step (2b): The network-server delivers $En-D_{js}$ to join-server.

Step (3): When receiving $En-D_{ed}$, the end-device performs the following sub-steps.

(3.1) calculates $k^{ns}_{ct} |D-Box_{new}| t^{RKU}_{nonce,ns} |\alpha|\ \text{DevEUI} = EDASS\left(NwkSKey_{old},\ D-Box_{old}, En-D_{ed}\right)$;
(3.2) takes the system time $t^{RKU}_{nonce,ed}$, and verifies the condition $t^{RKU}_{nonce,ed} - t^{RKU}_{nonce,ns} \leq \Delta t_1$, where Δt_1 is a predefined time difference including maximal transmission latency and the computational time of steps (1.3) to (1.6). If the verification failed, the end-device submits warning messages to network-server and join-server, and then stops the RKU stage; otherwise, it
(3.3) according to $DevEUI$, retrieves the corresponding $DevNonce$, calculates $k^{ed}_{ct} = SHA(t^{RKU}_{nonce,ns} \oplus \alpha \oplus DevNonce)$, and checks to see whether $k^{ed}_{ct} = k^{ns}_{ct}$ or not. If not, the end-device submits warning messages to network-server and join-server, and then stops the RKU stage; otherwise, it
(3.4) generates a random number β;
(3.5) calculates $En-Pr_{ed} = EDASS\left(AppSKey_{old},\ D-Box_{new}, \beta\ |\alpha \oplus DevNonce| t^{RKU}_{nonce,ed}\right)$;

Step (4): When receiving $En-D_{js}$ from network-server, join-server performs the following sub-steps.

(4.1) calculates $k_{ct}^{ns} |D-Box_{new}| t_{nonce,ns}^{RKU} |\alpha = EDASS\left(k_{ns-js},\ D-Box_{old},\ En-D_{ed}\right)$;

(4.2) takes the system time $t_{nonce,js}^{RKU}$, and verifies the condition $t_{nonce,js}^{RKU} - t_{nonce,ns}^{RKU} \leq \Delta t_2$, where Δt_2 is a predefined time difference including maximal transmission latency and the computational time of steps (1.3) to (1.6). If the verification failed, the join-server submits warning messages to network-server and end-device, and then stops the RKU stage; otherwise, it

(4.3) according to *DevEUI*, retrieves the corresponding *DevNonce*, calculates $k_{ct}^{js} = SHA(t_{nonce,ns}^{RKU} \oplus \alpha \oplus DevNonce)$, and checks to see whether $k_{ct}^{js} = k_{ct}^{ns}$ or not. If not, the join-server submits warning messages to network-server and end-device, and then stops the RKU stage; otherwise, it

(4.4) generates a random number γ;

(4.5) calculates $En-Pr_{js} = EDASS\left(AppSKey_{old},\ D-Box_{new},\ \gamma\ |\alpha \oplus DevNonce|\ t_{nonce,js}^{RKU}\right)$;

Step (5): The end-device sends $En-Pr_{ed}$ to join-server.

Step (6): The join-server sends $En-Pr_{js}$ to end-device.

Step (7): When receiving $En-Pr_{js}$ from join-server, the end-device performs the following sub-steps.

(7.1) calculates $\gamma\ |\alpha \oplus DevNonce|\ t_{nonce,js}^{RKU} = EDASS\left(AppSKey_{old},\ D-Box_{new},\ En-Pr_{js}\right)$;

(7.2) takes the system time $t_{nonce,ed}^{\prime RKU}$, and verifies the condition $t_{nonce,ed}^{\prime RKU} - t_{nonce,js}^{RKU} \leq \Delta t_3$, where Δt_3 is a predefined time difference including maximal transmission latency and the computational time of steps (4.3) to (4.5). If the verification failed, the end-device submits warning messages to network-server and join-server, and then stops the RKU stage; otherwise, it

(7.3) verifies $\alpha \oplus DevNonce$ is correct or not. If the value is not correct, the end-device submits warning messages to network-server and join-server, and then stops the RKU stage; or else, it

(7.4) computes new root keys $AppKey_{new} = (AppKey_{old} \oplus \beta) + (AppKey_{old} \oplus \gamma)$ and $NwkKey_{new} = (NwkKey_{old} \oplus \beta) + (NwkKey_{old} \oplus \gamma)$.

Step (8): When receiving $En-Pr_{ed}$ from end-device, the join-server performs the following sub-steps.

(8.1) calculates $\beta\ |\alpha|\ t_{nonce,ed}^{RKU} = EDASS\left(AppSKey_{old},\ D-Box_{new},\ En-Pr_{ed}\right)$;

(8.2) takes the system time $t_{nonce,js}^{\prime RKU}$, and verifies the condition $t_{nonce,js}^{\prime RKU} - t_{nonce,ed}^{RKU} \leq \Delta t_4$, where Δt_4 is a predefined time difference including maximal transmission latency and the computational time of steps (3.3) to (3.5). If the verification failed, the join-server submits warning messages to network-server and end-device, and then stops the RKU stage; otherwise, it

(8.3) verifies $\alpha \oplus DevNonce$ is correct or not. If the value is not correct, the join-server submits warning messages to network-server and end-device, and stops the RKU stage; or else, it

(8.4) computes new root keys which are the same with those equations in Step (7.4).

3.3 Stage 2: Session Key Update Procedure

In the SKU stage, once the join-server receives the session key update request sent by network-server, it generates an important parameter ϵ, and utilizes ϵ and new root keys generated in RKU stage to calculate new session keys, i.e., *AppSKey* and *NwkSKey*. Then the parameter ϵ is delivered to the end-device so that the end-device can generate the same session keys. Next, the end-device sends a new-session-key encrypted acknowledgement message to join-server, and the join-server confirms the acknowledgement message to ensure that the session keys generated by the end-device are the correct. After that, the join-server delivers network session key and application session key to network-server and application-server, respectively. Finally, both application-server and network-server send

encrypted messages to the end-device to guarantee new session keys' function. As shown in Fig. 4, in the SKU stage, there are 13 steps, which are described as follows.

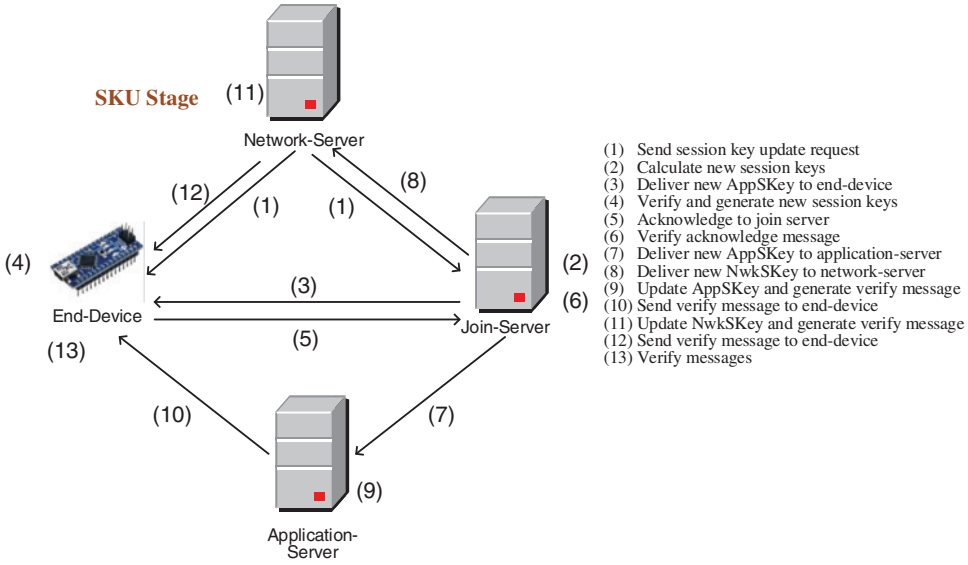

Figure 4: The SKU stage consists of 13 steps

Step (1): The network-server sends a SKU request and DevEUI to end-device and join-server.

Step (2): When receiving the SKU request, the join-server performs the following sub-steps.

(2.1) according to *DevEUI*, retrieves *DevNonce* (a 16-bit counter managed by end-device), *JoinEUI* (a global application ID in IEEE EUI64 address space that uniquely identifies the join-server), *AppKey* (one of the original root keys), and *NwkKey* (one of the original root keys) from its database;

(2.2) generates a 128-bit random number ϵ;

(2.3) fetches system time $t^{SKU}_{nonce,js}$ and calculate $S^{js}_{ct} = SHA\left(t^{SKU}_{nonce,js} \oplus \epsilon\right)$, where SHA stands for secure hash algorithm;

(2.4) calculates new session keys

$$AppSKey_{new} = EDASS(AppKey_{new}, D{-}Box_{new}, 0x02|\ \text{JoinNonce}\ |JoinEUI\ |\ \text{DevNonce}\ |\epsilon\ |\ S^{js}_{ct}|Pad_{16})$$

$$NwkSKey_{new} = EDASS(NwkKey_{new}, D{-}Box_{new},\ 0x01|\ \text{JoinNonce}\ |JoinEUI|\ \text{DevNonce}\ |\epsilon\ |\ S^{js}_{ct}|Pad_{16})$$

where Pad_{16} is appended zero to make the length of transmitted message satisfying a multiple of 16.

(2.5) calculates $En{-}K_{js2ed} = EDASS\left(AppSKey_{old},\ D{-}Box_{new},\ AppSKey_{new}\left|t^{SKU}_{nonce,js}\right|\ \text{DevEUI}\right)$;

Step (3): The join-server delivers $En{-}K_{js2ed}$ to end-device.

Step (4): When receiving $En{-}K_{js2ed}$, the end-device performs the following sub-steps.

(4.1) calculates $AppSKey_{new}\left|t^{SKU}_{nonce,js}\right|\ \text{DevEUI} = EDASS\left(AppSKey_{old},\ D{-}Box_{new},\ En{-}K_{js2ed}\right)$;

(4.2) obtains $JoinNonce\ |\ JoinEUI\ |\ DevNonce\ |\ \epsilon|\ S^{js}_{ct}$ by calculating $EDASS(AppKey_{new}, D-Box_{new},\ AppSKey_{new})$

(4.3) fetches system time $t^{SKU}_{nonce,ed}$ and verifies the condition $t^{SKU}_{nonce,ed} - t^{SKU}_{nonce,js} \leq \Delta t_5$, where Δt_5 is a predefined time difference including maximal transmission latency and the computational time

of steps (2.4) and (2.5). If the verification failed, the end-device submits warning messages to network-server and join-server, and then stops the SKU stage; otherwise, it

(4.4) calculates $S_{ct}^{ed} = SHA\left(t_{nonce,js}^{SKU} \oplus \epsilon\right)$ and checks to see whether $S_{ct}^{ed} = S_{ct}^{js}$ or not. If not, the end-device submits warning messages to network-server and join-server, and then stops the SKU stage; otherwise, it

(4.5) calculates new session keys with the equations same to those listed Step (2.4);

(4.6) calculates the acknowledgement message $Ack_{ed2js} = EDASS(AppSKey_{new},\ D{-}Box_{new},\ NwkSKey_{new}|t_{nonce,ed}^{SKU})$.

Step (5): The end-device sends the acknowledge message Ack_{ed2js} to join-server.

Step (6): Once receiving Ack_{ed2js}, the join-server performs the following sub-steps.

(6.1) decrypts Ack_{ed2js}, and retrieves $NwkSKey_{new}$ and $t_{nonce,ed}^{SKU}$;

(6.2) fetches system time $t_{nonce,js}^{'SKU}$ and verifies the condition $t_{nonce,js}^{'SKU} - t_{nonce,ed}^{SKU} \leq \Delta t_6$, where Δt_6 is a predefined time difference including maximal transmission latency and the computational time of steps (4.4) to (4.6). If the verification failed, the join-server submits warning messages to network-server and end-device, and then stops the SKU stage; otherwise, it

(6.3) checks to see whether $NwkSKey_{new}$ obtained from Step (6.1) are the same with Step (2.4) or not. If not, the join-server submits warning messages to network-server and end-device, and then stops the SKU stage; otherwise, it

(6.4) calculates $S_{ct}^{'js} = SHA\left(t_{nonce,js}^{'SKU} \oplus \epsilon\right)$

(6.5) calculates $En{-}K_{js2as} = EDASS(k_{js2ns},\ D{-}Box_{new},\ AppSKey_{new}|t_{nonce,js}^{'SKU}|\text{DevEUI}|\ t_{nonce,ed}^{SKU}|\epsilon|\ S_{ct}^{'js})$;

(6.6) calculates $En{-}K_{js2ns} = EDASS(k_{js2ns},\ D{-}Box_{new},\ NwkSKey_{new}|t_{nonce,js}^{'SKU}|\text{DevEUI}|\ t_{nonce,ed}^{SKU}|\epsilon|\ S_{ct}^{'js})$.

Step (7): Join-server sends $En{-}K_{js2as}$ to application-server.

Step (8): Join-server sends $En{-}K_{js2ns}$ to network-server.

Step (9): Once receiving $En{-}K_{js2as}$, the application-server performs the following sub-steps.

(9.1) decrypts $En{-}K_{js2as}$, and retrieves $AppSKey_{new}|t_{nonce,js}^{'SKU}|\text{DevEUI}|\ t_{nonce,ed}^{SKU}|\epsilon|\ S_{ct}^{'js}$;

(9.2) fetches system time $t_{nonce,as}^{SKU}$ and verifies the condition $t_{nonce,as}^{SKU} - t_{nonce,js}^{'SKU} \leq \Delta t_7$, where Δt_7 is a predefined time difference including maximal transmission latency and the computational time of steps (6.3) to (6.6). If the verification failed, the application-server submits warning messages to network-server, join-server, and end-device, and then stops the SKU stage; otherwise, it If not, it sends a warning message to network-server, and stops the SKU stage; otherwise, it

(9.3) calculates $S_{ct}^{'as} = SHA\left(t_{nonce,js}^{'SKU} \oplus \epsilon\right)$ and checks to see whether $S_{ct}^{'as} = S_{ct}^{'js}$ or not. If not, the application-server submits warning messages to network-server, join-server, and end-device, and then stops the RKU stage; otherwise, it

(9.4) updates session key $AppSKey_{new}$;

(9.5) calculates $En{-}K_{as2ed} = EDASS(AppSKey_{new},\ D{-}Box_{new}, \text{DevEUI}\ |t_{nonce,ed}^{SKU})$;

Step (10): Application-server sends $En{-}K_{as2ed}$ to end-device.

Step (11): On receiving $En{-}K_{js2ns}$ from join-server, the network-server performs the following sub-steps.

(11.1) decrypts $En{-}K_{js2ns}$, and retrieves $NwkSKey_{new}$, $t_{nonce,js}^{'SKU}$, DevEUI, $t_{nonce,ed}^{SKU}$, ϵ and $S_{ct}^{'js}$;

(11.2) fetches system time $t_{nonce,ns}^{SKU}$ and verifies the condition $t_{nonce,ns}^{SKU} - t_{nonce,js}^{'SKU} \leq \Delta t_8$, where Δt_8 is a predefined time difference including maximal transmission latency and the computational time of steps (6.3) to (6.6). If the verification failed, the network-server submits warning messages to join-server, application-server, and end-device, and then stops the SKU stage; otherwise, it

(11.3) calculates $S_{ct}^{'ns} = SHA\left(t_{nonce,js}^{'SKU} \oplus \epsilon\right)$ and checks to see whether $S_{ct}^{'ns} = S_{ct}^{'js}$ or not. If not, the network-server submits warning messages to other servers and end-device, and then stops the SKU stage; otherwise, it

(11.4) updates session keys $NwkSKey_{new}$

(11.5) calculates $En{-}K_{ns2ed} = EDASS(NwkSKey_{new},\ D{-}Box_{new}, \text{DevEUI}|t_{nonce,ed}^{SKU})$;

Step (12): Application-server sends $En{-}K_{ns2ed}$ to end-device.

Step (13): On receiving $En{-}K_{as2ed}$ from application-server, and $En{-}K_{ns2ed}$ from network-server, the end-device performs the following sub-steps.

(13.1) decrypts $En{-}K_{as2ed}$, obtains DevEUI and $t_{nonce,ed}^{SKU}$, and checks to see whether $t_{nonce,ed}^{SKU}$ is the same with that shown in Step (4.3) or not. If not, the end-device submits warning messages to all servers, and then stops the SKU stage; otherwise, it

(13.2) decrypts $En{-}K_{ns2ed}$, obtains DevEUI and $t_{nonce,ed}^{SKU}$, and checks to see whether $t_{nonce,ed}^{SKU}$ is the same with that listed in Step (4.3) or not. If not, the end-device submits warning messages to all servers, and then stops the SKU stage; otherwise, the SKU stage finishes.

4 Security Analyses

In this section, we analyze the security of THUS with Scyther tool [26] and discuss the security features of the THUS. The Scyther tool traces all parameters of the THUS, and checks whether these parameters are safe during transmission. The analyzed result is shown in Fig. 5, in which the important parameters are not threatened by attackers.

The other THUS security features are discussed as follows.

- **Mutual authentication** indicates that the data transmission parties can authenticate with each other. In the RKU stage of the THUS, both *DevEUI* and *DevNonce* are used to guarantee the authenticities of each communication pair. When an attacker lacks correct *DevNonce*, the correct k_{ct}^{ns} and $\alpha \oplus DevNonce$ cannot be calculated, and fails the checking in Steps (3.3), (4.3), (7.3), and (8.3). In the SKU stage, the *JoinNonce*, *JoinEUI*, and *DevNonce* are adopted to verify the identities of join-server, network-server, end-device, and application-server. Once an attacker is short of correct *JoinNonce*, *JoinEUI*, and *DevNonce*, the accurate $AppSKey_{new}$ and $NwkSKey_{new}$ cannot be calculated, and thus the Steps (4.6), (6.5), and (6.6) of SKU stage will generate wrong message, so that the checking in Steps (6.3), (9.3), and (11.3) will also be failed. Thus, the THUS equips mutual authentication feature.
- **Confidentiality** represents that the THUS has the ability to protect all important parameters. In the RKU stage and SKU stage of the THUS, all parameters become ciphertext by utilizing modified AES before transmission. In Steps (2a) and (2b) of RKU stage, the messages $D{-}Box_{new}$ and α are encrypted by using encryption keys $NwkSKey_{old}$ and k_{ns-js}, respectively. In Steps (5) and (6) of RKU stage, the messages are both encrypted by using $AppSKey_{old}$. In Steps (3) and (5) of SKU stage, the messages are encrypted by using $AppSKey_{old}$ and $AppSKey_{new}$, respectively. In Steps (7) and (10) of SKU stage, the messages are encrypted by using $AppSKey_{new}$, while in Steps (8) and (12) of SKU stage, the messages are encrypted by using $NwkSKey_{new}$. According to abovementioned steps, different messages and important parameters can then be protected by using different encryption keys.

Figure 5: The analyzed result of the proposed THUS

- **Message integrity** feature makes sure the transmitted messages not be forged. In the THUS, eight message integrity patterns, i.e., k_{ct}^{ns}, k_{ct}^{ed}, k_{ct}^{js}, S_{ct}^{js}, S_{ct}^{ed}, $S_{ct}^{'js}$, $S_{ct}^{'as}$, and $S_{ct}^{'ns}$ are generated to guarantee the integrity of transmitted message. When one of these patterns fails, the RKU or SKU stage is stopped. Consequentially, the THUS equips message integrity feature.
- **Resist replay attack** means some attacker captures some messages delivered by sender, duplicates those messages, and then retransmit them to receiver so as to pretend to be the legal sender. In the THUS, both RKU and SKU stages utilize time parameters (t_{nonce}) to resist replay attack.

Before data transmission, the time parameter is encrypted and embedded in the transmitted message. As shown in steps (3.2), (4.2), (7.2), and (8.2) of the RKU and steps (4.3), (6.2), (9.2), and (11.2) of the SKU, when receiver receives the message, the time parameter is utilized to guarantee the transmission time shorter than a predefine time limit. Once the actual transmission time is longer than the predefined time limit, it indicates that the message may send by illegal part, which means it may suffer replay attack. Accordingly, the THUS has the ability to prevent the replay attack.

- **Resist eavesdropping attack** indicates that an attacker captures massive messages and tries to obtain important information from them. In the THUS, parameters α, β, γ, ε, and D-Box are the most important parameters. The former three parameters are generated randomly and the D-Box is also produced by utilizing random box R-Box via DBG algorithm. All of these parameters and D-Box are encrypted by using EDASS before transmission. Since these parameters vary in every RKU stage and SKU stage, attackers cannot extract any parameters from the captured messages.

The comparisons between the proposed THUS and five previous works are listed in Tab. 2. As shown, the THUS equips not only root key update scheme, but also session key update scheme. By using modified AES algorithm, the THUS has the ability to provide high level security with efficient process.

Table 2: The comparisons between the THUS and five previous works

Works	Major contributions	Session key update	Root key update	Security features
[18]	Use trusted third party to deliver session keys.	V	X	confidentiality, integrity, authentication
[22]	Design a master device to manage root key.	X	V	no life time root key, eavesdropping attack resistance
[25]	Present a low power method to generate session keys.	V	X	known-key attack resistance, replay attack resistance, eavesdropping attack resistance
[20]	Address key updating, key generation, key backup, and key backward compatibility.	V	X	Known-key attack resistance, eavesdropping attack resistance, frequently key renew policy
[21]	Utilize elliptic curve cryptography to update LoRaWAN root key.	X	V	Forward and backward secrecy, man-in-the-middle attack resistance, message integrity and authentication

(Continued)

Table 2: Continued

Works	Major contributions	Session key update	Root key update	Security features
Proposed THUS	Use modified AES algorithm to update both LoRaWAN root keys and session keys.	V	V	Mutual authentication, confidentiality, message integrity, replay attack resistance, eavesdropping attack resistance

5 THUS Performance

Tab. 3 lists the operating time of the THUS, where T_r indicates random number generation time, T_{SHA} is the secure hash algorithm computational time, T_{AES} stands for AES computational time, T_{EDASS} represents the EDASS algorithm operation time, and T_b means the binary operation (addition and subtraction) time. According to the evaluation proposed in [24,25], T_{EDASS} is less than half of T_{AES}, and T_r and T_b can be ignored when compared with T_{AES}, T_{EDASS}, and T_{SHA}, thus the THUS with modified AES has the feature of high-efficiency when compared with traditional AES encryption method.

Table 3: The operating time of THUS with traditional AES and modified AES

Equipment	THUS with traditional AES	THUS with modified AES
Network-server	RKU: $T_r + T_{SHA} + 2T_{AES}$ SKU: $T_{SHA} + 2T_{AES} + T_b$	RKU: $T_r + T_{SHA} + 2T_{EDASS}$ SKU: $T_{SHA} + 2T_{EDASS} + T_b$
End-device	RKU: $T_r + T_{SHA} + 3T_{AES} + 4T_b$ SKU: $T_{SHA} + 7T_{AES} + T_b$	RKU: $T_r + T_{SHA} + 3T_{EDASS} + 4T_b$ SKU: $T_{SHA} + 7T_{EDASS} + T_b$
Join-server	RKU: $T_r + T_{SHA} + 3T_{AES} + 4T_b$ SKU: $T_r + T_{SHA} + 6T_{AES} + T_b$	RKU: $T_r + T_{SHA} + 3T_{EDASS} + 4T_b$ SKU: $T_r + T_{SHA} + 6T_{EDASS} + T_b$
Application-server	SKU: $T_{SHA} + 2T_{AES} + T_b$	SKU: $T_{SHA} + 2T_{EDASS} + T_b$

6 Conclusion and Future Studies

Secure data communication is a basic requirement for every IoT system. Although LoRaWAN utilizes AES cryptography to achieve the goal of data integrity and confidentiality, the encryption key management can be further improved. In this paper, the THUS is proposed to update LoRaWAN's root keys and session keys so that the LoRaWAN's security scheme can be enhanced. The modified AES algorithm with dynamic substitution-box is utilized in both RKU stage and SKU stage. According to the security discussion above, the proposed THUS has the features of mutual authentication, confidentiality and message integrity, and can resist replay and eavesdropping attacks.

In the future, the formal security verification for the RKU and SKU will be performed. Besides, the implementation of THUS on a FPAG (Field Programmable Gate Array) system will also be developed so that the real performance can be evaluated. These constitute our future studies.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] H. Boyes, B. Hallaq, J. Cunningham and T. Watson, "The industrial internet of things (IIoT): An analysis framework," *Computers in Industry*, vol. 101, pp. 1–12, 2018.

[2] G. J. Joyia, R. M. Liaqat, A. Farooq and S. Rehman, "Internet of medical things (IoMT): Applications, benefits and future challenges in healthcare domain," *Journal of Communications*, vol. 12, no. 4, pp. 240–247, 2017.

[3] K. L. Tsai, F. Y. Leu and I. You, "Residence energy control system based on wireless smart socket and IoT," *IEEE Access*, vol. 4, pp. 2885–2894, 2016.

[4] A. Kirimtat, O. Krejcar, A. Kertesz and M. F. Tasgetiren, "Future trends and current state of smart city concepts: A survey," *IEEE Access*, vol. 8, pp. 86448–86467, 2020.

[5] D. Flore, 3GPP Standards for the Internet-of-Thigs, GSMA MIoT, 2016.

[6] LoRaWAN, Accessed: Dec. 20, 2021. [Online]. Available: https://www.lora-alliance.org/.

[7] Sigfox, Accessed: Dec. 20, 2021. [Online]. Available: https://www.sigfox.com/.

[8] DASH7, Accessed: Dec. 20, 2021. [Online]. Available: https://dash7-alliance.org/.

[9] M. Alizadeh, K. Andersson and O. Schelén, "A survey of secure internet of things in relation to blockchain," *Journal of Internet Services and Information Security (JISIS)*, vol. 10, no. 3, pp. 47–75, 2020.

[10] H. Hui, X. An, H. Wang, W. Ju, H. Yang *et al.,* "Survey on blockchain for internet of things," *Journal of Internet Services and Information Security (JISIS)*, vol. 9, no. 2, pp. 1–30, 2019.

[11] National Inst of Standards and Technology Gaithersburg MD, "Announcing the Advanced Encryption Standard (AES)," Federal Information Processing Standards Publication 197, United States National Institute of Standards and Technology (NIST), 2001.

[12] K. L. Tsai, F. Y. Leu, L. L. Hung and C. Y. Ko, "Secure session key generation method for LoRaWAN servers," *IEEE Access*, vol. 8, pp. 54631–54640, 2020.

[13] H. Noura, T. Hatoum, O. Salman, J. P. Yaacoub and A. Chehab, "LoRaWAN security survey: Issues, threats and possible mitigation techniques," *Internet of Things*, vol. 12, pp. 100303, 2020.

[14] K. L. Tsai, F. Y. Leu, I. You, S. W. Chang, S. J. Hu *et al.,* "Low-power AES data encryption architecture for a LoRaWAN," *IEEE Access*, vol. 7, pp. 146348–146357, 2019.

[15] I. You, S. Kwon, G. Choudhary, V. Sharma and J. T. Seo, "An enhanced LoRaWAN security protocol for privacy preservation in IoT with a case study on a smart factory-enabled parking system," *Sensors*, vol. 18, no. 6, pp. 1888, 2018.

[16] LoRa Alliance Technical Committee, LoRaWAN Backend Interfaces 1.0 Specification, LoRa Alliance, 2017.

[17] LoRa Alliance Technical Committee, "LoRaWAN 1.1 specification," LoRa Alliance, 2017.

[18] S. Naoui, M. E. Elhdhili and L. A. Saidane, "Trusted third party based key management for enhancing LoRaWAN security," in *Proc. of IEEE/ACS 14th Int. Conf. on Computer Systems and Applications (AICCSA)*, Hammamet, Tunisia, pp. 1306–1313, 2017.

[19] M. Eldefrawy, I. Butun, N. Pereira and M. Gidlund, "Formal security analysis of LoRaWAN," *Computer Networks*, vol. 148, pp. 328–339, 2019.

[20] X. Chen, M. Lech and L. Wang, "A complete key management scheme for LoRaWAN v1.1," *Sensors*, vol. 21, no. 9, Article: 2962, pp. 1–20, 2021.

[21] J. Xing, L. Hou, K. Zhang and K. Zheng, "An improved secure key management scheme for LoRa system," in *Proc. of IEEE 19th Int. Conf. on Communication Technology*, Xi'an, China, pp. 296–301, 2019.

[22] T. C. M. Dönmez and E. Nigussie, "Key management through delegation for LoRaWAN based healthcare monitoring systems," in *Proc. of 13th Int. Symp. on Medical Information and Communication Technology (ISMICT)*, Oslo, Norway, pp. 1–6, 2019.

[23] J. Han and J. Wang, "An enhanced key management scheme for LoRaWAN," *Cryptography*, vol. 2, no. 4, article 34, pp. 1–12, 2018.
[24] J. J. Liu, Y. L. Huang, F. Y. Leu, X. Y. Pan and L. R. Chen, "Generating dynamic box by using an input string," in *Proc. of Int. Symp. on Mobile Internet Security*, Jeju Island, Korea, pp. 1–13, 2017.
[25] K. L. Tsai, Y. L. Huang, F. Y. Leu, I. You, Y. L. Huang *et al.,* "AES-128 based secure low power communication for LoRaWAN IoT environments," *IEEE Access*, vol. 6, pp. 45325–45334, 2018.
[26] The Scyther Tool. Accessed: Jan. 25, 2022. [Online]. Available: https://people.cispa.io/cas.cremers/scyther/.