cryptography

MDPI

Article

An Enhanced Key Management Scheme for LoRaWAN

Jialuo Han * and Jidong Wang

School of Engineering, RMIT University, Melbourne 3000, Australia; jidong.wang@rmit.edu.au
* Correspondence: jialuo.han@student.rmit.edu.au; Tel.: +61-451-898-776

Received: 21 August 2018; Accepted: 29 October 2018; Published: 2 November 2018

Abstract: The LoRaWAN is one of the new low-power wide-area network (LPWAN) standards applied to Internet of Things (IoT) technology. The key features of LPWAN are its low power consumption and long-range coverage. The LoRaWAN 1.1 specification includes a basic security scheme. However, this scheme could be further improved in the aspect of key management. In this paper, LoRaWAN 1.1 security is reviewed, and enhanced LoRaWAN security with a root key update scheme is proposed. The root key update will make cryptoanalysis of security keys in LoRaWAN more difficult. The analysis and simulation show that the proposed root key update scheme requires fewer computing resources compared with other key derivation schemes, including the scheme used in the LoRaWAN session key update. The results also show the key generated in the proposed scheme has a high degree of randomness, which is a basic requirement for a security key.

Keywords: LoRaWAN security; Internet of Things; key management

1. Introduction

In recent years, Internet of Things (IoT) technology has emerged as a trend in wireless communication applications. The IoT could enable billions of physical entities and sensors to be deployed in the context of smart homes, smart cities, and smart industries. These entities are connected to the Internet for monitoring and control types of applications [1]. The current wireless communication protocols, such as Bluetooth, Wi-Fi, and 3G/4G, support high data rates but cannot meet both the requirements of low power consumption and wide area coverage. The LoRaWAN is a low-power wide-area network (LPWAN) protocol, which eliminates the complexity of deployment, while supporting both the features of power efficiency and long-range transmission [2]. It is optimized for environments that require a low data rate (0.3 kbps–50 kbps), low power consumption and wide coverage area. LoRa is a radio modulation technology by using the chirp spread spectrum (CSS) found in industrial, scientific, and medical (ISM) unlicensed bands. Its frequency bands vary from region to region, and include EU868, EU433, US915 and AS430 [3]. The LoRaWAN 1.1, a Media Access Control (MAC) -layer protocol, is implemented as a star-of-star topology to regulate LoRa devices. A LoRaWAN comprises End Devices (ED), Radio Gateways (GW), Network Servers (NS), and Application Servers: an ED is responsible for transmitting the collected data from the sensor; a GW is a middle bridge which forwards packets between the ED and NS; an NS is at the center of the star topology, linking with all the nodes based on LoRa MAC; and, an AS handles the application layer payloads with data encryption/decryption applied [4]. There are two ways to enable an ED to join a LoRaWAN network. They are the Over the Air (OTA) and Activation by Personalization (ABP) procedures. OTA performs an exchange of two MAC messages (i.e., Join-Request and Join-Accept) between the ED and NS for activation. A group of session parameters will be generated and exchanged in LoRa devices for securing the session.

The original LoRaWAN Specification 1.0 has a basic security mechanism. LoRaWAN Specification 1.1 [5], released at the end of 2017, contains a significant improvement relating to key management and data confidentiality. Two 128-bit pre-shared root keys (i.e., NwkKey, AppKey) are stored in both the ED and the Join Server (JS). These are used to derive network and application session keys, respectively, which are used to ensure payload integrity in the MAC layer and confidentiality in the application layer. The network session keys include FNwkSIntKey, SNwkSIntKey, and NwkSEncKey. The first two of these are used for calculating a Message Integrity Code (MIC) for up/downlink messages by using Advanced Encryption Standard- Cipher-based Message Authentication Code (AES-CMAC). NwkSEncKey is responsible for the encryption/decryption of MAC commands. The application session key, AppSKey is used to encrypt application payload with AES in counter mode. Thus, the network session key and the application session key are only known and handled in NS and AS, respectively, to prevent malicious access to the application payload during transmission.

However, the LoRaWAN Specification 1.1 still has some weaknesses in root key management. The lack of LoRaWAN session key management has been addressed by using Ephemeral Diffie-Hellman Over COSE (EDHOC) in the paper by Sanchez-Iborra [6]. The issue of forwarding secrecy to support end-to-end security between the LoRa device and the LoRa server is addressed by You [7]. Furthermore, symmetric encryption implementation means that root keys should be stored at two places, i.e., in the ED and JS. The EDs are most vulnerable to various attacks, such as side channel analysis, as they are remotely allocated in most applications. Hackers can detect fluctuations in power consumption from the LoRa transceiver during AES encryption, which can help to derive the key used [8]. Once root keys are stolen, the ED will be compromised for its lifetime as the root keys are fixed. To overcome this problem, a root key update procedure can be adopted to enhance the security of root key handling.

The contributions of this paper are: (1) A review of LoRaWAN 1.1 security flaws relating to key management and data confidentiality; (2) analysis of a lightweight key update scheme for improving LoRaWAN 1.1 key management; (3) proposal of a lightweight key management scheme based on the Rabbit cipher embedded in a two-step Key Derivation Function (KDF); and, (4) comparison of the results of the proposed scheme with other main key derivation methods.

The rest of the paper is organized as follows. Firstly, a review of the LoRaWAN 1.1 security scheme is presented, with a focus on key management. The discussion that follows reveals flaws in that scheme. Then, a new key management solution is proposed. Analysis and discussion of the new scheme are presented with some comparisons with the existing scheme. The conclusion summarizes the improvements of the new scheme.

2. LoRaWAN Security Review

The LoRaWAN 1.1 security scheme is shown in Figure 1. The protocol provides packet confidentiality by symmetric encryption between ED and NS, and between ED and AS. Root keys are stored in the entities that are responsible for the derivation of session keys [8]. In LoRaWAN 1.1, two AES-128 pre-shared keys, NwkKey and AppKey, are stored in the non-volatile memory of ED and JS before activation. Compared with LoRaWAN 1.0, NwkKey eliminates the risk of unauthorized access and theft of the payload prior to AS because NS could be a third-party device for which it is neither necessary, nor desirable, to know AppSKey [8,9]. Moreover, detailed network session keys, which are responsible for message integrity checking and MAC payload encryption, have been introduced in LoRaWAN 1.1. The rest of this section provides a brief introduction to these LoRaWAN 1.1 features, i.e., key generation in the join procedure, key distribution, and payload encryption methods.

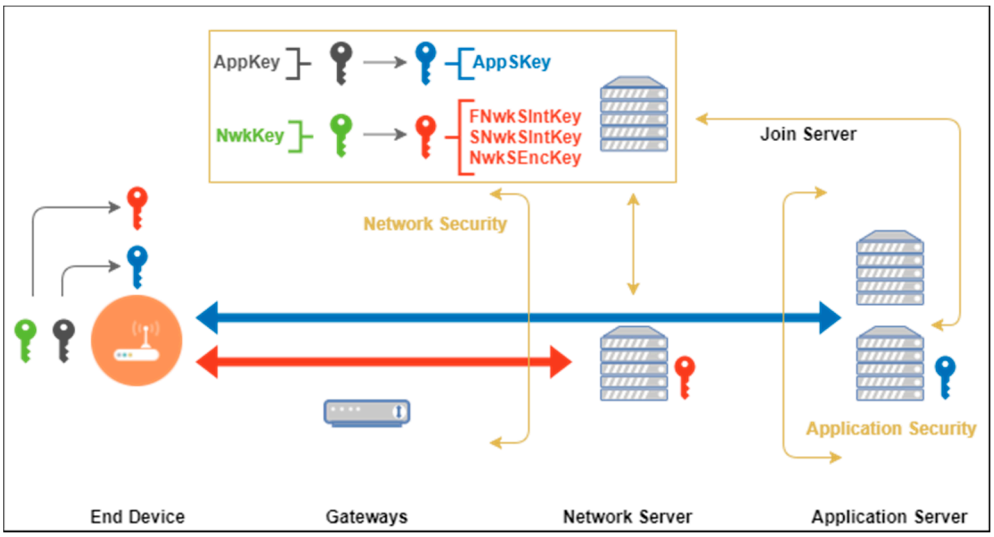

Figure 1. LoRaWAN 1.1 Key Distribution.

2.1. Key Derivation and Distribution

The derivation of session keys is closely related to the join procedure, which is carried out by the entities of ED, NS, and JS. The unencrypted Join-Request message is signed with NwkKey, which will be initialized by ED, forwarded to NS and handled by JS. The packet structure of the Join-Request is shown in Figure 2. It contains JoinEUI and DevEUI, of 8-bytes each, and 2-byte DevNonce. JoinEUI is a global application ID and DevEUI is a global end device ID. DevNonce has been changed to a counter value instead of a pseudo-random value, and cannot be reused with the same JoinEUI, in order to prevent replay attack [10,11]. The NS will reply with a Join-Accept message to ED once the Join-Request is accepted. The Join-Accept message structure is shown in Figure 3. As JoinNonce value is a crucial parameter involved in key derivation, the Join-Accept is encrypted by AES in Electronic Codebook (ECB) mode using NwkKey. The MIC is produced for the whole encrypted content using NwkKey. JoinNonce is also modified to a counter value to prevent replay attack. In the join procedure, the symmetric keys are not transmitted over the air, but JoinNonce is used to generate session keys on both sides.

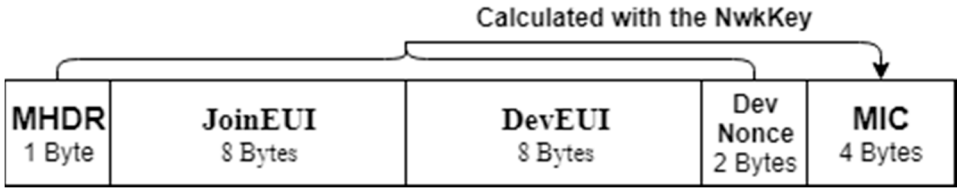

Figure 2. LoRaWAN 1.1 Join-Request message.

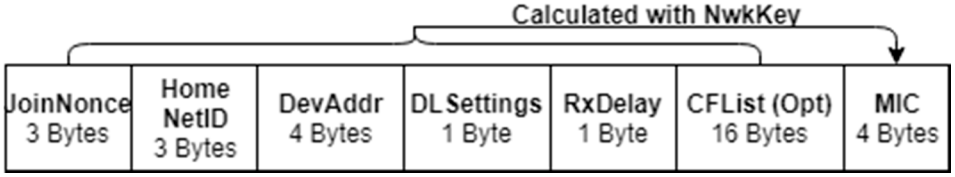

Figure 3. LoRaWAN 1.1 Join-Accept message.

In LoRaWAN 1.1, three network session keys are defined: FNwkSIntKey, SNwkSIntKey, and NwkSEncKey. These session keys are responsible for the up/downlink data message integrity check and encryption of the MAC payload separately. The application session key is used for application

payload encryption between ED and AS. Session key derivation has adopted AES-128 in ECB mode and requires the parameters included in the Join-Accept message with root keys NwkKey and AppKey, which are stored in an ED and the JS through their lifetimes. The three parameters used for session key derivation are JoinEUI, DevNonce, and JoinNonce. Both nonce values are counter values generated in each join procedure, which are unencrypted. In total, four session keys are defined. The derivation and the use of each session key is explained in the following:

- Forwarding Network session integrity key (FNwkSIntKey): One of the three network session keys of 128-bit length. It is used to calculate the partial MIC (the second two bytes) of uplink data messages. NS would verify the MIC when it receives the message from ED. The key derivation can be represented as follows:

$$\text{FNwkSIntKey} = \text{aes128\_encrypt}\big(\text{NwkKey}, 0\text{x}01|\text{JoinNonce}|\text{JoinEUI}|\text{DevNonce}|\text{pad}_{16}\big) \quad (1)$$

- Serving Network session integrity key (SNwkSIntKey): Also a 128-bit network session key that is used to calculate the first two bytes of the MIC of uplink data messages. It is also used to calculate the entire 32-bit MIC of the downlink data message. The MIC of the downlink data message would be verified in ED to secure message integrity. Its key derivation can be represented as follows:

$$\text{SNwkSIntKey} = \text{aes128\_encrypt}\big(\text{NwkKey}, 0\text{x}03|\text{JoinNonce}|\text{JoinEUI}|\text{DevNonce}|\text{pad}_{16}\big) \quad (2)$$

- Network session encryption key (NwkSEncKey): The third network session key for encryption/decryption of MAC commands for uplink and downlink. ED and NS are the only LoRa devices that hold this data. Its derivation is as follows:

$$\text{NwkSEncKey} = \text{aes128\_encrypted}\big(\text{NwkKey}, 0\text{x}04|\text{JoinNonce}|\text{JoinEUI}|\text{DevNonce}|\text{pad}_{16}\big) \quad (3)$$

- Application session key (AppSKey): A 128-bit application session key as defined in LoRaWAN 1.0, i.e., it is used for the application payload's encryption/decryption between ED and AS. It is derived as follows:

$$\text{AppSKey} = \text{aes128\_encrypt}\big(\text{AppKey}, 0\text{x}02|\text{JoinNonce}|\text{JoinEUI}|\text{DevNonce}|\text{pad}_{16}\big) \quad (4)$$

2.2. Data Confidentiality

LoRa devices are generally deployed in remote locations. Data confidentiality is often an important consideration as the radio transmission can be intercepted. With the new keys defined in LoRaWAN 1.1, the data encryption and authentication have been enhanced. The following section describes the operations of data encryption and message authentication in the new release.

The MAC frame payload is encrypted using AES-128 in Counter mode (CTR), and the MIC is calculated after the encryption. The keystream generation and payload encryption/decryption are described as follows:

$$\begin{aligned}
&i = 1 \ldots k \\
&\text{where :} \\
&k = ceil(len(FRMPayload)/16) \\
&Ai = (0x01|0x00|Dir|DevAddr|FCntUP \text{ or } \text{NFC}ntDown \text{ or AFCntDown}|0x00|i) \\
&Si = aes128_encry(K, Ai),\ K \in \{AppSKey,\ NwkSEncKey\} \\
&S = S1|S \ldots|Sk \\
&[FRMPayload] = (S1|S2 \ldots|Sk) \oplus (FRMPayload|\ pad_{16})
\end{aligned} \quad (5)$$

In the above formulation, 'Dir' indicates the direction of uplink/downlink message (uplink = 0x00, downlink = 0x01), and 'Cnt' is a 32-bit counter that can be either an uplink or downlink counter in the calculation. The downlink counter in v1.0 has been replaced with two counters in LoRaWAN 1.1, NFCntDown and AFCntDown, to differentiate between the counters applied in NS (Port 0) and AS (Port 1-233). Both counters will be initialized to 0 at session start and incremented in each transmission in a synchronized manner. 'DevAddr' is a static value presented as a device address obtained from the NS and delivered in the Join-Accept message. The encryption/decryption method is AES in Counter mode and the ciphertext is formed as the result of an XOR operation on the keystream [12] and the padded payload.

To provide the integrity of the ciphertext, a 4-byte MIC is used as the authentication tag between ED and Servers, and is computed with AES-CMAC, which has been listed in Table 1. In LoRaWAN 1.1, SNwkSIntKey and FNwkSIntKey have been introduced to calculate the MIC of uplink/downlink data separately; this differs from LoRaWAN 1.0, in which only one session key applies for both uplink and downlink MIC. MIC calculation contains one network session key that indicates different types of message, an encrypted payload, and a 16-prefix block represented as B_0 and B_1. After calculation, the result will be truncated to a 4-byte MIC and appended to the corresponding message, and then checked once it arrives at NS/ED. If the check fails, the message will be discarded. This is an efficient method to detect bit-flipping and replay attacks. The MIC for the data message is computed with $\text{msg} = MHDR \mid FHDR \mid FPort \mid FRMPayload$ as follows:

Table 1. LoRaWAN 1.1 Message Integrity Code (MIC) Calculation.

MIC Types	Calculation Process	Parameters
Downlink Frames $MIC = cmac[03]$	$cmac_{downlink} = aes128\_cmac(SNwkSIntKey,\ B_0 \mid msg)$	Where: $\boldsymbol{B_0} = 0x49(1) \parallel ConfFCnt(2) \parallel 0x00(2) \parallel Dir(1) \parallel DevAddr(4) \parallel AFCntDwn/NFCntDwn(4) \parallel 0x00(1) \parallel len(msg)(1)$
Uplink Frames $MIC = cmacS[01] \mid cmacF[01]$	$cmacS_{uplink} = aes128\_cmac(SNwkSIntKey,\ B_1 \mid msg)$, $cmacF_{uplink} = aes128\_cmac(fNwkSIntKey,\ B_0 \mid msg)$	Where: $\boldsymbol{B_0} = 0x49(1) \parallel 0x00(4) \parallel Dir(1) \parallel DevAddr(4) \parallel FCntUp(4) \parallel 0x00(1) \parallel len(msg)(1)$ $\boldsymbol{B_1} = 0x49(1) \parallel ConfFCnt(2) \parallel TxDr(1) \parallel TxCh(1) \parallel Dir(1) \parallel DevAddr(4) \parallel AFCntUp(4) \parallel 0x00(1) \parallel len(msg)(1)$

2.3. Security Key Management

An outline of LoRaWAN 1.1 key derivation, distribution and data confidentiality is presented above. The security scheme focuses on confidentiality and integrity of messages in an IoT scenario. In this section, the discussion relates to possible attack of the network with OTA activation. The security assets to be protected are most important parameters in LoRaWAN network, and form the core of the security of network.

In the OTA procedure, all session keys are generated from two static root keys, NwkKey and AppKey, which are pre-shared between ED and JS. Some of the contents are not protected in the transmission. For example, the Join-Request message is not encrypted. Its contents, including JoinEUI and DevNonce, can be sniffed by an attacker. However, JoinNonce is a confidential parameter transmitted with the Join-Accept message. These are common values applied to derive session keys

with one of the root keys. As noted, session and root keys are key assets, and the LoRaWAN network will be compromised if knowledge of these keys is obtained by an adversary. Thus, according to William Stallings, frequent key exchanges are usually desirable to limit the amount of data compromised if an attacker learns the key [13]. Hence, it is necessary to consider a mechanism that updates session contents and root keys over time. Such updates can lower the risk of node capture attacks and side-channel attacks, such as those via Electromagnetic radiation (EM) emissions or power consumption, to which the network can be susceptible because the LoRa device is generally remotely located [14–16]. A session key update scheme has been introduced in LoRaWAN 1.1, which is enabled and configured by the MAC commands. However, static root keys, which cannot be updated during the device's lifetime, are used to derive all session keys. In this scenario, static root keys can cause a serious issue with key leakage because, as described, the remaining parameters of session key derivation can be easily obtained. A root key update scheme with low power consumption should be introduced to enhance LoRaWAN security according to various studies of key management, which recommend that the key should be updated periodically [17,18].

3. Lightweight Root Key Update Scheme

Based on the analysis of current LoRaWAN key management, LoRa ED is the most vulnerable part of the LoRaWAN network because it is remotely located, and stores all of the pre-shared root keys that are used for session keys derivation. In LoRaWAN 1.1, the session key update scheme uses Rejoin-Request commands to trigger an update by the same key derivation method. However, the root key is static through its lifetime, and an attacker has sufficient time to obtain knowledge of session key derivation, such as by sniffing the keying material over the air. This risk can be reduced by periodically updating root keys. The current session key derivation method—i.e., AES-ECB mode—is not suitable for root key derivation because AES-ECB is vulnerable to pattern analysis with sufficient knowledge and has higher power consumption. It is expected that root key updates should be less frequent than session key updates, and that their frequency should be set in a reasonable range because frequent root key updates in a resource-constrained node would adversely affect its lifespan. Rabbit is a high-efficiency synchronous stream cipher based on simple arithmetic [19] and other basic operations. The output from its keystream generator is a 128-bit string in each round, and it has suitable randomness. Its attributes are compatible with LoRaWAN properties, namely, low power consumption, lightweight computing, and randomized output. According to the National Institute of Standards and Technology (NIST) Special Publication SP 800, a two-step KDF is an approved cryptographic algorithm which consists of a pseudo-random number generator (PNG) that is used to generate the basic building block [20]. The Rabbit stream cipher is used as an efficient PNG in this scenario. In the following section, the process of a Rabbit-based two-step KDF is presented for the LoRaWAN root key update scheme.

3.1. Security Key Management

A two-step KDF involves two phases, which consist of an extractor and an expander. The algorithm is shown in Table 2. Both phases are based on the PNG of the Rabbit stream cipher to obtain keystreams. As Figure 4 shows, the randomness extractor is used to take a root key, k_a, and a series of shared contexts, c, which are assumed to be non-uniformly random values. Session keys can be considered to be a part of contexts because they will be immediately renewed once the root key is updated. Both k_a and c are used to generate a pseudo-random value [21]. The key expander takes the output of the extractor and another root key as the input. The output of the expander, a sequence of 128 bits, will be the new root key, and is a pseudo-random number whose randomness is close to a uniform distribution (as shown in a later section). The operation block diagram and model of the two-step KDF proposed in the LoRaWAN root key update scheme is:

$$KDF(k_a, k_b, c) = Expander\{Extractor(k_a, c), k_b\} \quad (6)$$

Table 2. Rabbit-based KDF algorithm.

$for\ i = 0 : L,\ do$	$for\ i = 0 : L,\ do$
$if\ i == 0$	$if\ i == 0$
$k_a = B_i[0 : v];$	$k_{kdk} = B_i[0 : v];$
$c = B_i[v : v + w] = 0x00;$	$k_b = B_i[v : v + w];$
$[k_a, c] = keystream[v + w] \oplus B_{i+1}$	$keystream[v + w];$
$else\ if\ 0 < i < L$	$[k_{kdk},\ k_b] = keystream[v + w]$
$k_a = B_i[0 : v];$	$i = i + 1;$
$c = B_i[v : v + w];$	$Output\ a\ 128\ bits\ strings\ as\ {k_a}'$
$keystream[v + w];$	$else\ if\ i == 1$
$keystream[v + w] \oplus B_{i+1}$	$k_{kdk} = B_i[0 : v];$
$i = i + 1;$	$k_b = B_i[v : v + w];$
$else\ i = L$	$keystream[v + w];$
$k_a = B_i[0 : v];$	$Output\ a\ 128\ bits\ strings\ as\ {k_b}'$
$c = B_i[v : v + w];$	$else$
$Output\ a\ 128\ bits\ string,\ k_{kdk}[0 : s];$	end

Where :
$[v + w] = 128\ bits$
$pl :\ the\ total\ length\ of\ keying\ material$
$B_i :\ the\ i^{th}\ v + w\ bits\ block\ of\ keying\ material$
$k_{kdk} :\ Key\ derivation\ key$
$k_a, k_b :\ old\ root\ keys$
$k'_a, k'_b :\ new\ root\ keys$
$v, w, s :\ root\ key\ block\ size,\ contexts\ block\ size,\ k_{kdk}\ size$
$c : Contexts$
$L = \frac{pl}{v+w} - 1 : the\ total\ number\ of\ blocks$

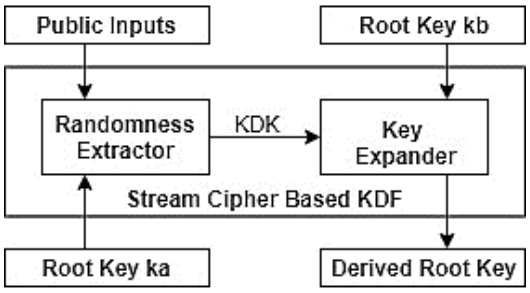

Figure 4. Two-step Key Derivation Function (KDF) structure.

3.1.1. Randomness Extractor

As described above, the randomness extractor is based on the PNG of the Rabbit stream cipher. As shown in Figure 5, it has two steps: an initialization process and keystream generation. In the first step, the total length of the keying material is *pl*, which contains a root key and a series of contexts. The root key is divided *into* v-bit long blocks and the contexts are divided into w-bit long blocks (where $v + w = 128$) for the Rabbit stream cipher. If the length of the block is less than $v + w$ bits, it will be padded with zeros. In the second step, the number of iterations in the state function has an impact on the randomness of the stream generated. Its selection should balance the resulting randomness and computing requirements. The 128-bit pseudo-random keystream from the PNG will be XORed with the next block of the keying material and the result will be fed back to the next loop of the PNG. The pseudo-random keystream comprises zeros in the initial loop. When it reaches the last block of keying material, a 128-bit keystream, k_{kdk}, will be extracted, which will be the input of the key expander. The process of the extractor is shown in Figure 5.

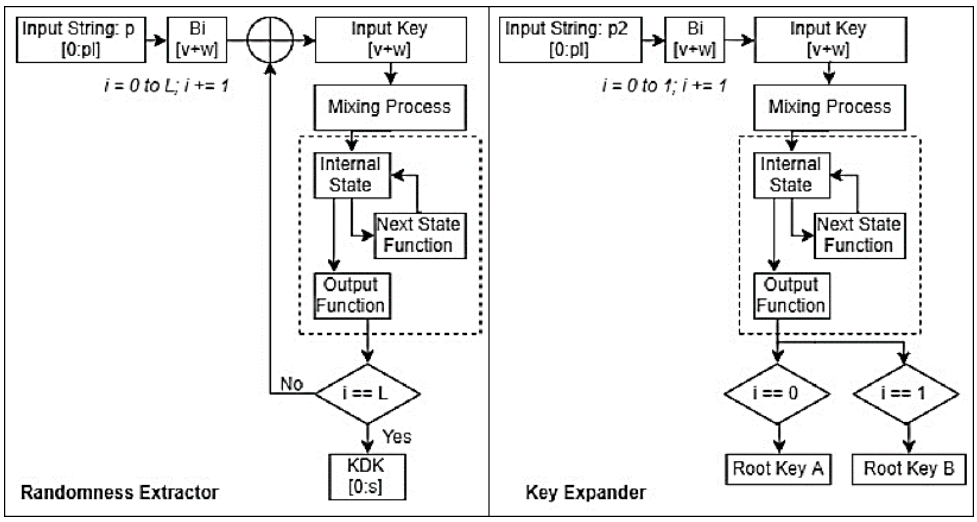

Figure 5. Rabbit-based KDF Structure.

3.1.2. Key Expander

In the second phase of the KDF, the key expander also uses the PNG of the Rabbit stream cipher. The keying material in this step is specified as p2 with a length of pl. It includes a 128-bit Key Derivation Key, k_{kdk}, and a root key, k_b. Another root key is divided into blocks each with a length of v, and k_{kdk} is divided into blocks each with a length of w, where $v + w = 128$ bits. The total length of keying material is 256-bit in this scenario, and two loops of the Rabbit stream cipher will be executed. Two new 128-bit root keys will be generated in the first and second loops, in which each loop takes $B_i = v + w$ as the input. If the keying material length is greater than 256 bits, more loops can be carried out to generate more keys. The right part of Figure 5 illustrates the second phase of the proposed Rabbit stream cipher-based KDF.

3.2. Root Key Distribution Scheme

A LoRa ED is distributed remotely in most situations. The root key update scheme does not require physical access to the ED, as the update is carried out using the existing key setting between ED and JS. A LoRa ED and the JS are the only two entities that hold the knowledge of the root keys. Based on the two-step KDF, transmission of the updated root keys between LoRa EDs and the JS is not required. A set of MAC commands can be designed to negotiate the root key update process that is shown in Figure 6. It includes passing the context parameters used for keying materials from JS to the EDs. A root key request is initiated from the NS. The update can be carried out regularly or triggered by certain events. The update request from the NS includes an update identifier and a nonce value. When an ED receives an update request from the NS via a downlink MAC command, it confirms with the NS if it can verify the request command. Then, NS sends a command to JS which holds the ED's root keys to request the root key update. The JS will confirm with the NS once it verifies the request command. Then both JS and the ED go through the same KDF process separately. Once the ED completes the KDF process, it will send the JS a new Key Ready command containing a nonce with a MIC generated using the newly generated root keys. When JS receives the command, it will use its own newly generated root keys to verify the MIC. If it is verified, the JS will send a new Key Ready confirmation command to NS, and NS will pass the confirmation command to the ED. Once the ED has the confirmation, it will start a new Join-Request process as specified in LoRaWAN 1.1. Once the ED has rejoined the network, the root key update is complete.

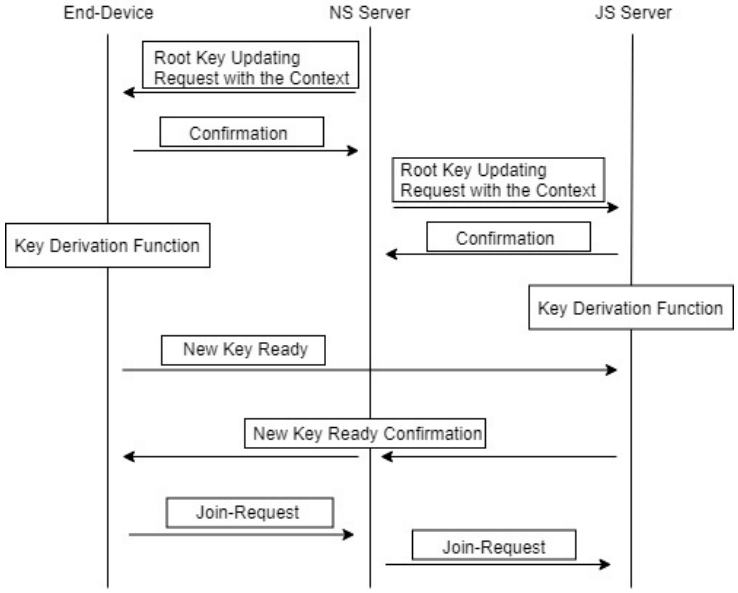

Figure 6. Root Key update scheme.

The root key update should take power consumption into the consideration. A too-frequent update will consume more power, and may not be necessary. Two main factors to be considered in the update are the lifetime of the root keys (i.e., the duration they are valid) and the length of messages transmitted with the valid root keys. If the root keys have been used for a long period or too many session keys have been generated using the root keys, then the root keys should be updated.

4. Analysis of Root Key Update Scheme

In this proposed scheme, the Rabbit stream cipher-based KDF is the core of the LoRaWAN root key update. Compared with current KDFs, such as HKDF [22], which is a key derivation function based on hash message authentication, and AES-CMAC-based KDF [23,24], it has a lower computing-weight of key derivation and does not have the cryptographical weaknesses revealed with the Rabbit stream cipher [25]. The two-step KDF [23] includes randomness extractor and key expansion phases, is used to obtain the Key Derivation Key from shared secrets and derives updated keys. Hence, the Rabbit stream cipher is used in a two-step KDF for the LoRaWAN root key update scheme. For the scheme evaluation, a PC with the following specification was used: Intel (R) Core (TM) i7-7700K CPU 4.2 GHz 4.2 GHz with 8 GB RAM, running 64-bit Windows 10. The performance of three key derivation methods—Rabbit-based KDF, HKDF (SHA1), and LoRaWAN 1.1 session key derivation (AES128-ECB)—are analyzed in separate experiments. It is meaningful to compare these common key derivation methods with our proposed scheme in the same testing environment because the main comparison relates to the efficiency of the key derivation method. The time taken for these key derivation methods on the PC means similar results will be achieved on LoRa devices. From the results it can be reckoned that the faster the derivation, the lower the power consumption on the LoRa device.

The key derivation computing times are compared in Table 3 for the three methods. The computing time is for 10,000 repeats of each method. They are obtained using a built-in Windows C++ function QueryPerformanceCounter(). As shown in the table, it is obvious that the Rabbit-based KDF is the fastest method, and that the AES-ECB mode used in LoRaWAN 1.1 session key derivation is the slowest method among the three. Furthermore, the LoRaWAN 1.1 session key derivation generates only one 128-bit key each time, and takes the longest time in generating a 128-bit key.

Table 3. Performance of key derivation methods.

Algorithm	*pl*	*kl*	*ul*	*t*
Rabbit Based KDF	80	16	32	0.0092
Hash Based KDF	80	80	32	0.1215
AES-ECB mode	16	16	16	0.1313

pl: Size of keying material, *kl*: size of input key, *ul*: size of updated key, *t*: average execution time (ms).

Compared with the LoRaWAN 1.1 session key update period, the root key update period should be longer to make the root update meaningful. The session key update period should depend on the amount of message transferred. If enough messages have been transferred in one session, a new session key is necessary to counter attack cryptoanalysis. LoRaWAN 1.1 recommends the following session key update period: $period_{session} = 2^{T+10}$ s. Thus, the root key update period should be longer than session key update period, that is, at least 2^{11} s.

It is apparent that frequent root key updates will enhance LoRaWAN security. However, the more frequent the update, the higher the computing resource required. As the Rabbit-based KDF dramatically reduces the key derivation time, it makes frequent updates of the root key more affordable for LoRa EDs with constrained power [24].

Key randomness is a basic and essential factor to measure the performance of a KDF. Key randomness of the Rabbit-based KDF is evaluated on 1000 key samples, which are used to calculate the standard deviation and probability density function (PDF) for checking the randomness of the updated key. The 128-bit updated key is split into sixteen 8-bit blocks. The analysis is based on the randomness of blocks. Figure 7a shows the PDF of updated keys in each block; the range of each block is from 0 to 255. The PDF is close to a uniform distribution, which leads to less correlation among the blocks and ensures key randomness. The standard deviation of the 16 octets representing the 16 blocks of the derived key is shown in Figure 7b with the average value (mean). It shows that the block values are spread evenly over a wide range and that root key randomness is guaranteed in the proposed Rabbit-based KDF.

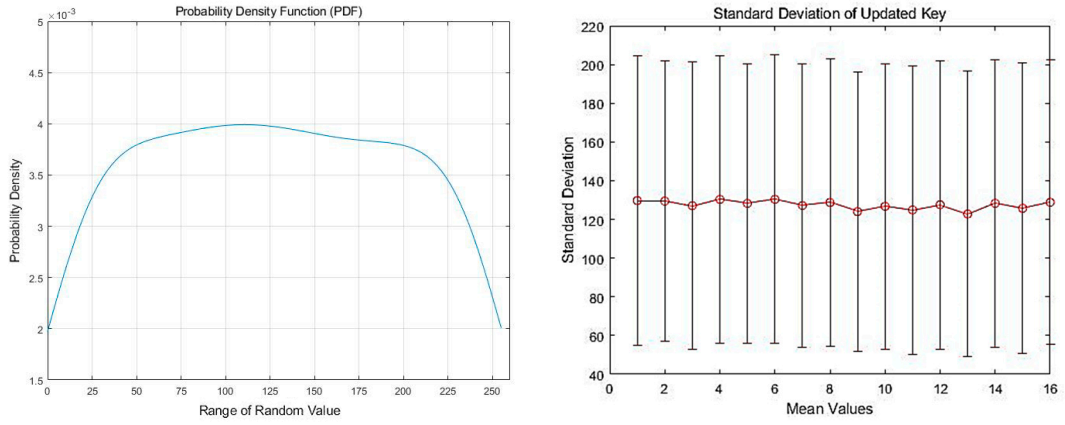

Figure 7. (**a**) Probability density function of updated key; (**b**) standard deviation of updated key.

5. Conclusions

In this paper, LoRaWAN security is reviewed, and some vulnerabilities of its key update are identified. A root key update scheme is proposed to strengthen the security of session key derivation. The proposed scheme applies a Rabbit stream cipher-based KDF to update the root key. The full root key update procedure is provided, and its evaluation is carried out on a Windows PC. In this paper, our work is compared with LoRaWAN session key derivation and hash-based KDF methods. The proposed scheme has the highest computing efficiency and offers suitable randomness of the

generated updated key. As root key updating will enhance overall LoRaWAN security, the proposed KDF can be a candidate for the introduction of root key updates in a future LoRaWAN release.

Author Contributions: Formal Analysis, J.H.; Methodology, J.H.; Supervision, J.W.; Writing—Original Draft, J.H.; Writing—Review & Editing, J.H. and J.W.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Naoui, S.; Elhdhili, M.E.; Saidane, L.A. Enhancing the security of the IoT LoraWAN architecture. In Proceedings of the 2016 International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN), Paris, France, 22–24 November 2016; pp. 1–7.
2. *A Technical Overview of LoRa and LoRaWAN*, 1st ed.; LoRa Alliance: Fremont, CA, USA, 2015.
3. Semtech. What Is LoRa? | Semtech LoRa Technology | Semtech. 2018. Available online: https://www.semtech.com/lora/what-is-lora (accessed on 6 October 2017).
4. Sornin, N.; Yegin, A. LoRaWAN Backend Interfaces 1.0 Specification. LoRa Alliance Standard Specification. 2017. Available online: www.lora-alliance.org (accessed on 20 March 2018).
5. Sornin, N.; Yegin, A. LoRa Specification 1.1. LoRa Alliance Standard Specification. 2017. Available online: www.lora-alliance.org (accessed on 20 March 2018).
6. Sanchez-Iborra, R.; Sánchez-Gómez, J.; Pérez, S.; Fernández, P.; Santa, J.; Hernández-Ramos, J.; Skarmeta, A. Enhancing LoRaWAN Security through a Lightweight and Authenticated Key Management Approach. *Sensors* **2018**, *18*, 1833. [CrossRef] [PubMed]
7. You, I.; Kwon, S.; Choudhary, G.; Sharma, V.; Seo, J.T. An Enhanced LoRaWAN Security Protocol for Privacy Preservation in IoT with a Case Study on a Smart Factory-Enabled Parking System. *Sensors* **2018**, *18*, 1888. [CrossRef] [PubMed]
8. Miller, R. *LoRa Security: Building a Secure LoRa Solution*, 1st ed.; MWR Labs Whitepaper; MWR Labs: London, UK, 2016.
9. Roman, R.; Alcaraz, C.; Lopez, J.; Sklavos, N. Key management systems for sensor networks in the context of the Internet of Things. *Comput. Electr. Eng.* **2011**, *37*, 147–159. [CrossRef]
10. SeungJae, N.; DongYeop, H.; WoonSeob, S.; Ki-Hyung, K. Scenario and countermeasure for replay attack using join request messages in LoRaWAN. In Proceedings of the 2017 International Conference on Information Networking (ICOIN), Da Nang, Vietnam, 11–13 January 2017; pp. 718–720.
11. Kim, J.; Song, J. A simple and efficient replay attack prevention scheme for LoRaWAN. In Proceedings of the 2017 the 7th International Conference on Communication and Network Security, Tokyo, Japan, 24–26 November 2017; pp. 32–36.
12. Gildas Avoine, L.F. *Rescuing LoRaWAN 1.0*; INSA Rennes, France; CNRS: Paris, France, 2016.
13. Stallings, W.; Brown, L. *Computer Security: Principles and Practice*, 2nd ed.; Pearson: Boston, MA, USA, 2012.
14. Zhang, Z.-K.; Cho, M.C.Y.; Shieh, S. Emerging security threats and countermeasures in IoT. In Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security (ASIA CCS'15), Singapore, 14–17 April 2015; ACM: New York, NY, USA, 2015; pp. 1–6.
15. Hossain, M.M.; Fotouhi, M.; Hasan, R. Towardsan Analysis of Security Issues, Challenges, and Open Problems in the Internet of Tings. In Proceedings of the IEEE World Congress on Services (SERVICES 2015), New York, NY, USA, 27 June–2 July 2015; pp. 21–28.
16. Zhao, K.; Ge, L. A survey on the internet of things security. In Proceedings of the 9th International Conference on Computational Intelligence and Security (CIS 2013), Emei Mountain, China, 14–15 December 2013; pp. 663–667.
17. He, J.; Zhang, X.; Wei, Q. EDDK: Energy-efficient distributed deterministic key management for wireless sensor networks. *EURASIP J. Wirel. Commun. Netw.* **2011**, *2011*, 765143.
18. Barker, E.; Barker, W.; Burr, W.; Polk, W.; Smid, M. *Recommendation for Key Management Part 1: General (Revision 4)*; NIST Special Publication: Washington, DC, USA, 2016.
19. RFC 4503—A Description of the Rabbit Stream Cipher Algorithm. IETF Tools. 2018. Available online: https://tools.ietf.org/html/rfc4503 (accessed on 23 October 2018).

20. Che, L. *NIST Special Publication 800-108*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2009.
21. Krawczyk, H. Cryptographic Extraction and Key Derivation: The HKDF Scheme. In *Advances in Cryptology—Proceedings of the 30th International Cryptology Conference (CRYPTO 2010), Santa Barbara, CA, USA, 15–19 August 2010*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 631–648.
22. Chen, L. *SP 800-56C. Recommendation for Key Derivation through Extraction-then-Expansion*; National Institute of Standards & Technology: Gaithersburg, MD, USA, 2011.
23. Dworkin, M.J. *SP 800-38B. Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*; National Institute of Standards & Technology: Gaithersburg, MD, USA, 2005.
24. Boesgaard, M.; Vesterager, M.; Zenner, E. The Rabbit Stream Cipher. In *New Stream Cipher Designs*; Matthew, R., Olivier, B., Eds.; Springer: Heidelberg, Germany, 2008; pp. 69–83.
25. Suárez-Albela, M.; Fernández-Caramés, T.M.; Fraga-Lamas, P.; Castedo, L. A practical evaluation of a high-security energy-efficient gateway for IoT fog computing applications. *Sensors* **2017**, *17*, 1978. [CrossRef] [PubMed]