A Novel Session Key Update Scheme for LoRaWAN

HAYATI, Nur, WINDARTA, Susila, SURYANEGARA, Muhammad, PRANGGONO, Bernardi <http://orcid.org/0000-0002-2992-697X> and RAMLI, Kalamullah

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 2 July 2022, accepted 11 August 2022, date of publication 19 August 2022, date of current version 30 August 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3200397

RESEARCH ARTICLE

A Novel Session Key Update Scheme for LoRaWAN

NUR HAYATI[1], (Member, IEEE), SUSILA WINDARTA[1], (Member, IEEE), MUHAMMAD SURYANEGARA[1], (Senior Member, IEEE), BERNARDI PRANGGONO[2], (Senior Member, IEEE), AND KALAMULLAH RAMLI[1], (Member, IEEE)

[1]Department of Electrical Engineering, Faculty of Engineering, Universitas Indonesia, Depok 16424, Indonesia
[2]Department of Engineering and Mathematics, Sheffield Hallam University, Sheffield S1 1WB, U.K.

Corresponding author: Kalamullah Ramli (kalamullah.ramli@ui.ac.id)

This research is partly supported by Universitas Indonesia through the Hibah Publikasi Terindeks Internasional (PUTI) Q1 Scheme under Contract NKB-509/UN2.RST/HKP.05.00/2022, of which Prof. Dr-Ing. Kalamullah Ramli is the corresponding author. Ms. Hayati is supported in her PhD study by Beasiswa Unggulan Dosen Indonesia Dalam Negeri (BUDI-DN), Lembaga Pengelola Dana Pendidikan (LPDP), cooperation of the Ministry of Research and Higher Education and the Ministry of Finance of the Republic of Indonesia.

ABSTRACT This paper proposes a novel Long-range Wide Area Network (LoRaWAN) session key updating scheme to enhance the security of LoRaWAN with cost-effective communication that provides a unique key for each communication session. The scheme consists of three sequential stages, i.e., initialization, keying material preparation, and key updating, on the basis of the truncated Photon-256 algorithm with updatable keying materials. These stages are structured by a set of novel communication protocols. To prove the uniqueness of the key, we validated its sequence bit randomness using the NIST 800-22 and ENT statistical test suites. The validation results show that the key passes all test parameters. Subsequently, the communication protocols were validated by using Scyther tools. We proved that these protocols ensure the security of the LoRaWAN key update scheme and guarantee that active interception does not occur. The analysis was performed by focusing on the security features of data confidentiality, integrity protection, mutual authentication, perfect forward secrecy, and replay attack resistance. Finally, a formal security analysis using GNY logic indicated that the overall security goals are achieved. The proposed scheme's performance was evaluated in terms of computational cost, communication cost, and storage. The computational cost needed by the scheme is very small, indicating that there is no additional burden on the backend system. The communication cost requires less traffic than previous solutions, yet it offers more robust security for LoRaWAN by producing a new key in every communication session. The scheme needs insignificant additional storage that is considered negligible.

INDEX TERMS LoRaWAN security, session key update, key management, secure protocol, truncated Photon-256.

I. INTRODUCTION

The Long-range Wide Area Network (LoRaWAN) is a new standard for Low Power Wide Area (LPWA) in the Internet of Things (IoT) that was established through Recommendation ITU-T Y.4480 [1], [2], [3]. LoRaWAN has been widely adopted by the industrial IoT [4], [5], [6] as well as academic research in various fields, such as smart cities [7], [8], [9] and healthcare [10], [11], [12]. The operation of LoRaWAN guarantees data security by dealing with two parts of cryptographic key management, i.e., the root key [13], [14], [15], [16] and the session key [17], [18], [19], [20]. The root key is the LoRaWAN principal key used to derive all other cryptographic keys [21], and the session key is a derivation key used to secure communication and payload transmission [22].

Our previous work in [16] proposed a novel scheme for root key update management operating on the basis of CTR_AES DRBG 128-bit. This scheme regularly changes

The associate editor coordinating the review of this manuscript and approving it for publication was Giovanni Pau.

the LoRaWAN root key from static to dynamic. As our previous work addressed the first issue of root key management, this paper reports work on the second issue, i.e., session key management.

Several studies have highlighted that LoRaWAN session key management has a problem, i.e., applying the same session key to secure multiple communication sessions [17], [18], [19], [20]. Key repetition leads to data leakage when it is compromised [14], especially since the LoRaWAN session key's function is to secure its communication and data payload. Moreover, NIST recommends that the session key should be applied only once in every communication or should be unique to each session [23].

The schemes for solving this problem that were proposed in [17], [18], [19], and [20] have limitations. These limitations include the high extra communication cost incurred by the entities involved; additionally, these works do not provide a complete security validation test. Details of these limitations are discussed in Section II-B.

We propose a novel LoRaWAN session key updating scheme that provides a unique key for each communication session. The scheme works on the basis of a truncated Photon-256 algorithm with updatable keying materials. We simulate the scheme by focusing on back-end system keying material preparation and the key updating procedure, and we validate the simulation results regarding the bit sequence randomness by employing the NIST 800-22 and ENT statistical test suites. Subsequently, we validate the proposed communication protocol's security utilizing Scyther tools; carry out a series of security analyses, both formal and informal; and examine the performance analysis of the proposed scheme.

The main contribution of this research is the development of a novel LoRaWAN session key updating scheme that uses a specific method and protocols:

- A method of updating the session key by applying the truncated Photon-256 algorithm with updatable keying material, which provides a unique session key for each communication session.
- A set of protocols that support secure and cost-effective communication. The protocols renew keying material between the end device, join server, network server, and application server.

The remaining sections of the paper are structured as follows: Section II explains the underlying theory of the LoRaWAN session keys and related work. Section III describes the proposed novel secure session key update scheme for LoRaWAN, and Section IV presents the validation results and discussion. The last section, Section V, concludes the research.

II. UNDERLYING THEORY AND RELATED WORKS

A. LoRaWAN SESSION KEYS

The LoRaWAN protocol is equipped with the advanced encryption standard (AES) algorithm to protect data integrity and confidentiality during transmission [24]. The AES algorithm has two different keys, i.e., the session key and root key [18], [22], [25]. The root key is a symmetric key applied to secure data transmission between the end device and the join server. The session key is a symmetric key used to preserve the security of the communication protocol and data payload transmitted between the end device, network server, and application server [25], [26].

The LoRaWAN session keys comprise application and network session keys. The application session key guarantees data confidentiality by encrypting the payload exchanged between the end device and the application server. The network session keys ensure communication security between the end device and the network server.

LoRaWAN differentiates its network session keys into three types: the forwarding network session integrity key (*FNwk*SIntKey), serving network session integrity key (*SNwkSIntKey*), and network session encryption key (*NwkSEncKey*). *FNwkSIntKey* is a network session key that is used to calculate the message integrity code (*MIC*)value of the uplink data sent by the end device. The function of *SNwkSIntKey* is to verify the*MIC* value for all downlink messages and ensure data integrity. *NwkSEncKey* encrypts the LoRaWAN data payload throughout transmission from the end device to the server and vice versa [22], [27]. An illustration of the LoRaWAN session keys is shown in Fig. 1.

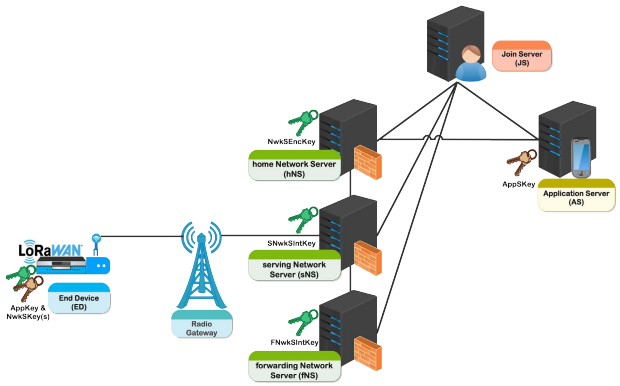

FIGURE 1. The LoRaWAN session keys.

LoRaWAN derives the session keys from its root key through a key derivation function (KDF) [22], [28], [29], [30], [31]. The end device and join server run the KDF with the *AES_ECB* algorithm and input the root key as the KDF cryptographic key parameter [22]. Although the LoRaWAN session keys are dynamic, the value rarely changes. By default, the session key of LoRaWAN only changes in the rejoin procedure when the end device loses its communication session with the server [22], [28], [29], [30], [31]. Thus, assuming that LoRaWAN maintains its communication session for a long time, an identical session key is used during this time. LoRaWAN applies the same key to secure the communication session and encrypt the payload during a specific period. In this case, the LoRaWAN session key becomes insecure, and it exceeds the crypto period standard [23], [32]. This condition also leads to data leakage and does not comply with the security standards recommended by NIST [23], [32]. The

NIST recommendation specifies that the session key must be different in each communication session [23]. Therefore, some studies have attempted to resolve this issue by proposing session key update schemes [17], [18], [19], [20].

B. RELATED WORK

Researchers have conducted studies to address the LoRaWAN key management issue [17], [33]. To improve security key management, researchers have proposed a method of updating the key, as the fundamental problem of LoRaWAN's key management is that the key has a constant value for a long time. Since LoRaWAN has a root key and session key, some studies have focused on updating the root key [13], [14], [15], [16], while others have focused on the session key [17], [18], [19], [20]. The goal of the proposed solution is to improve LoRaWAN security.

Chen *et al.* [17] examined the entire LoRaWAN key management process. They proposed a session key update algorithm using the modified Rabbit PRNG algorithm. However, they did not provide standard validation to test the randomness of the new session key. The proposed method did not include a communication protocol to support the key updating scheme.

The research conducted by Tsai *et al.* [18] proposed SeLPC, a scheme for updating session keys with power optimization in the AES algorithm. SeLPC's power consumption optimization was carried out by reducing the rounds of the AES algorithm. To maintain the cryptographic keys' security level, they used a Dynamic Box instead of an S-Box, which was updated every k days. The session key was updated every k days by the network server. Every k days, devices that were clustered as one group received the new session key. This work solved this LoRaWAN problem and saved power consumption. However, this work did not provide randomness validation for the bit sequences. Since the proposed solution reduces the half-cycle and changes the S-Box of the AES algorithm, a randomness test of the output is necessary to confirm that the bit sequence output resulting from the modification satisfies the cryptographic key security requirement.

Xia *et al.* [19] proposed a session key update scheme to tighten the security of meter-reading systems based on LoRaWAN. It added a trusted key distribution server (KDS) that functions as a key management center. The center manages the session time and updates the session key for each smart meter-reading LoRaWAN node. The KDS controls the key with a particular expiration period. Thus, each time the session key expires during its lifecycle, the KDS communicates to the end device and the network server to update the session key. This solution dynamically provides a new session key with an unpredictable schedule. The session key is frequently updated over a certain period, but the value remains the same until it expires. In a condition in which the new session key is renewed for each data transmission, this scheme increases the communication traffic.

Research to improve the security of LoRaWAN cryptographic keys was also performed by Tsai *et al.* in [20]. The study proposed the low-power AES data encryption architecture (LPADA) as a session key update scheme to improve LoRaWAN's security key management with minimum power consumption. LPADA employs ultralow-power content addressable memory to design the Sbox AES algorithm for LoRaWAN. Thus, the AES algorithm is more energy efficient in updating the session key. However, the analysis of this study focuses more on energy consumption, neglecting the randomness and communication protocol tests.

Other research on session key update was conducted by Tsai *et al.* in [34]. This research mainly focused on generating and updating the session keys used among LoRaWAN servers. The update was not applied to the session key used for securing session communication and encrypting the payload. The proposed elliptic curve algorithm employs a 163-bit key length, yet the applied key size is less than the recommended cryptographic key length for the elliptic curve [35]. The relevant NIST recommendation states that during 2019–2030, the minimum cryptographic key size for the elliptic curve algorithm is 224 bits [23].

These previous works solved the LoRaWAN session key problem, yet they have limitations. Thus, our study proposes a solution and addresses their weaknesses. We propose a secure session key update scheme based on the Photon-256 algorithm. This research improves the methods of Chen *et al.* [38] by adding communication protocols to support the scheme.

Unlike Tsai *et al.* in [18], the new session key result of the update scheme is validated by two standardized NIST 800-22 Rev. 1a [36], [37] and ENT statistical test suites [38], [39]. This validation is an essential factor in proving the randomness value of the new session key. The new session key bit sequences that pass this validation satisfy the cryptographic key security requirement [23].

Furthermore, the research of Xia *et al.* [19] solved the LoRaWAN problem but demanded heavy traffic communications. We address this issue of intense communication by proposing a different solution that reduces the traffic. Instead of generating and delivering the new session key value directly, we have the server provide component of the keying material.

Regarding the protocol, this paper examines informal and formal security analysis to address the limitation that previous studies [19], [20] only examined informal analysis. We provide validation of the proposed communication protocols using Scyther tools, as was done by Tsai *et al.* in [34]. This validation proves the security of the protocol's design and ensures that an active attacker is unable to intercept communication so that the key's secrecy is guaranteed during transmission.

III. PROPOSED SESSION KEY UPDATE SCHEME FOR LORAWAN

Based on the LoRaWAN architecture, our proposed session key update scheme involves four entities: the end device (ED), join server (JS), network server (NS), and application server (AS). We exclude the radio gateway because the

scheme does not add particular tasks to the gateway. To simplify the explanation, we summarize the abbreviations of the involved entities in Table 1.

TABLE 1. Abbreviations of LoRaWAN entities used in the paper.

Abbreviation	LoRaWAN Entity
ED	End Device
JS	Join Server
NS	Network Server
AS	Application Server

A. GENERAL ARCHITECTURE

In our proposed scheme, the session key update algorithm is executed individually on three different entities, i.e., the ED, NS, and AS. The ED updates all required session keys, while the AS updates only the application session key. We set the NS involved in this scheme as the home Network Server (hNS). Therefore, the session key related to the serving Network Server (sNS) and forwarding Network Server (fNS) is updated by the home Network Server; later in this paper, we call it simply the NS.

We set the JS serves as a key generation center that is supported by a random bit generator (RBG) module [40] that provides and shares keying materials and attributes. The session keying material generated by the RBG module is a 128-bit pseudorandom bit sequence. The 128-bit pseudorandom bit sequence is then divided into two parts. The 64 most significant bits (MSBs) are assigned as the value of the master password for the network session key. Then, the remaining 64 least significant bits (LSBs) are assigned as the value of the master password for the application session key. Other entities (the ED, NS, and AS) then use the master password value as keying material's component.

The general architecture design of the proposed session key update scheme is displayed in Fig. 2. According to the design, pairs of entities exchange communication protocols to request and deliver session keying material components. There are three protocols exchanged between the pair ED-JS, and an end-to-end symmetric encryption algorithm protects these protocols. Then, three protocols are exchanged between the JS-NS and JS-AS pairs. The asymmetric encryption algorithm protects these protocols.

1) COMMUNICATION PROTOCOL EXCHANGED BETWEEN THE ED AND JS

In our scheme, the communication protocol sent by the ED to the JS is defined as a *New_Rejoin-request*. To secure its communication, the *New_Rejoin request* is protected by the MIC, where the MIC calculation utilizes the AES-CMAC 128-bit algorithm with an updatable LoRaWAN root key [16]. The *New_Rejoin request* initializes the session key update process to request keying materials and other attributes. We design the ED to send *New_Rejoin-request* communications on a periodic basis, such as once a week or every n days as scheduled [16]. To respond to such a request, the JS replies by sending a *New_Rejoin-response* that contains keying material

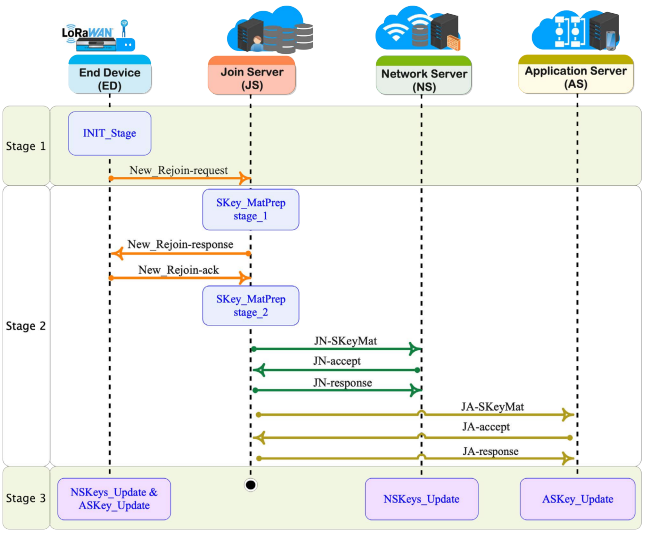

FIGURE 2. General architecture of the proposed session key update scheme for LoRaWANs.

components and attributes needed by the ED to process the session key update. The *New_Rejoin-response* is encrypted by AES-ECB 128-bit and is also equipped with a MIC. In the last step, to confirm and acknowledge the received keying materials, the ED sends *New_Rejoin-ack.* These communication protocol exchanges trigger the JS to communicate with the NS and AS concurrently.

2) COMMUNICATION PROTOCOLS EXCHANGED BETWEEN THE JS AND NS AND THE JS AND AS

We distinguish the communication protocol terms that are used between the JS-NS and JS-AS pairs, although we assume the communications occur concurrently. The communication protocols used among servers, the JS, NS and AS, are encrypted using the elliptic curve cryptography (ECC) algorithm [34], [41], [42], [43]. In the scheme, we assume that the JS-NS and JS-AS pairs have exchanged their public keys, which can be used directly in the session key update scheme.

The JS sends the *JN-SKeyMat* protocol to the NS containing the network session keying material and attributes needed by the NS. The NS sends a confirmation to the JS by sending *JN-accept*. Then, the JS responds by sending back *NonceNS* over the *JN-response*, ensuring that the NS has received the network session keying materials.

At the same time, the JS sends *JA-SKeyMat* to the AS, which contains the attributes and application session keying material needed by the AS. The AS then replies to the JS with *JA-accept*. Last, the JS sends *JA-response* to ensure that the AS has received the information used for updating the key.

3) THE ECC ALGORITHM SETUP TO SECURE COMMUNICATION BETWEEN JS-NS AND JS-AS PAIRS

The ECC algorithm used for securing communication between JS-NS and JS-AS pairs is defined as follows: In this research, the ECC algorithm utilizes Elliptic Curve25519 with a 256-bit key length [41], [42], [43]. We select Elliptic

Curve25519 because of its security performance and efficiency. The prime fields minimize ECC security problems, and the calculation of the EC points is fast [42]. Meanwhile, a 256-bit key length is chosen to comply with the minimum security key size for prolonged use beyond 2030 [35], [43]. The protocol messages are converted into points on an elliptic curve. The initial setup for ECC is as follows:

a: GLOBAL PUBLIC PARAMETERS

- *Curve*25519.
- Select a base point G in *Curve*25519 at random, the order of which is a large positive integer n.

b: KEY GENERATION

- Each user chooses a private key $n_{user} < n$ and generates a public key $P_{user} = n_{user} \times G$.

c: ENCRYPTION

1. To encrypt a message m, encode m to point $P_m = (x, y)$ in *Curve*25519.
2. The user chooses a random positive integer k and calculates a pair of points as the ciphertext C_m, as follows: $C_m = \{k \times G, P_m + k \times P_{recipient}\}$.

d: DECRYPTION

1. To decrypt the ciphertext C_m, the recipient calculates
   - $P_m + k \times P_{recipient} - n_{recipient} \times k \times G = P_m + k \times n_{recipient} \times G\text{-}n_{recipient} \times k \times G = P_m$.
2. P_m is decoded to the original message.

In this paper, the recipients are the JS, NS, and AS. In addition, a digital signature used in the scheme is *Curve*25519 with a private key to encrypt the digest value.

4) SESSION KEY UPDATE ALGORITHM

In the proposed scheme, we utilize the Photon-256 algorithm with 128 truncated outputs as the session key update algorithm. Photon-256 produces a hash code output that has a 224-bit length. The result is then truncated to 128 bits to adjust to the required length of the session key. We select the Photon algorithm because it is a lightweight algorithm suitable for an IoT end device [44], [45]. Since the computation of session key updates involves a LoRaWAN end device, a lightweight algorithm is necessary [44], [46]. In addition to being lightweight, the Photon algorithm has been standardized by the International Standard ISO/IEC 29192-5 [47]. Additionally, the truncated Photon-256 algorithm was chosen because it complies with the ISO/IEC 29192-1 security standard for lightweight cryptography [47], and the NIST recommendations, in which the hash value length for 2019-2030 is 224 bits [35], [43].

In our scheme, the session key is updated regularly based on a single communication session. The ED sends the payload in every scheduled period using a different session key.

To reduce the transmission costs incurred by the ED and the communication traffic among the entities, the ED only requests keying material once a week or as defined according to needs, every n days.

TABLE 2. Terms, notation, and definitions used in the study.

Term	Definition
INIT_Stage	The initialization stage of the session key update scheme in the ED.
SKEY_MatPrep	The session keying material preparation stage, which occurs on the JS. It comprises two substages: *Skey_MatPrep stage_1* and *Skey_MatPrep stage_2*.
NSKey_Update	The network session key update process, executed on the ED and NS.
ASKey_Update	The application session key update process, executed on the ED and AS.
MPNet	The master password of the network session keys. This is part of the session keying material used as the input parameter of the network session key update. It is sent only to the ED and NS.
MPApp	The master password of the application session key. This is part of the session keying material used as the input parameter of the application session key update and is sent only to the ED and AS.
Ts	ED timestamp
Ts'	JS timestamp
ΔT	Timestamp deviation between the JS and ED; $\Delta T = T' - T$.
MIC_{EJ}	Message integrity code (MIC) generated by the ED to protect a message sent to the JS.
MIC_{JE}	Message integrity code (MIC) generated by the JS to protect a message sent to the ED.
$NSKeyMat$	Network session keying materials used to update the network session keys.
$ASKeyMat$	Application session keying materials used to update the application session key.
AppID	Application server identifier
NonceJS	Nonce generated by the JS
NonceNS	Nonce generated by the NS
NonceAS	Nonce generated by the AS
Hash	Recommended hash function used to create and calculate the JS digital signature.
JSIntKey, JSEncKey	Standard end-to-end symmetric keys between the ED and JS defined by the LoRaWAN specification.
P_{JS}, and P_{NS}	Public key of the JS and public key of the NS.
K_{JS}, *and* K_{NS},	Private key of the JS and private key of the NS.

B. SESSION KEY UPDATE SCHEME

We describe three sequential stages to carry out the session key update scheme, as shown in Fig. 2. The first stage, *INIT_Stage*, is defined as an initialization process to request keying material for new sessions, which occurs in the ED. The second stage, *Skey_MatPrep*, is defined as a session's keying material preparation stage, which is processed on the JS. Skey_MatPrep has two substages: *Skey_MatPrep stage_1* and *Skey_MatPrep stage_2*. The third stage is the session key update procedure, comprising *NSKey_Update* and *ASKey_Update*. *NSKey_Update* is executed on the ED and NS, while *ASKey_Update* is executed on the ED and AS. The terms, notation, and definitions used in this proposed scheme are defined in Table 2.

All three stages are interrelated, but each has a different role. Fig. 3 illustrates the functional diagram of the proposed

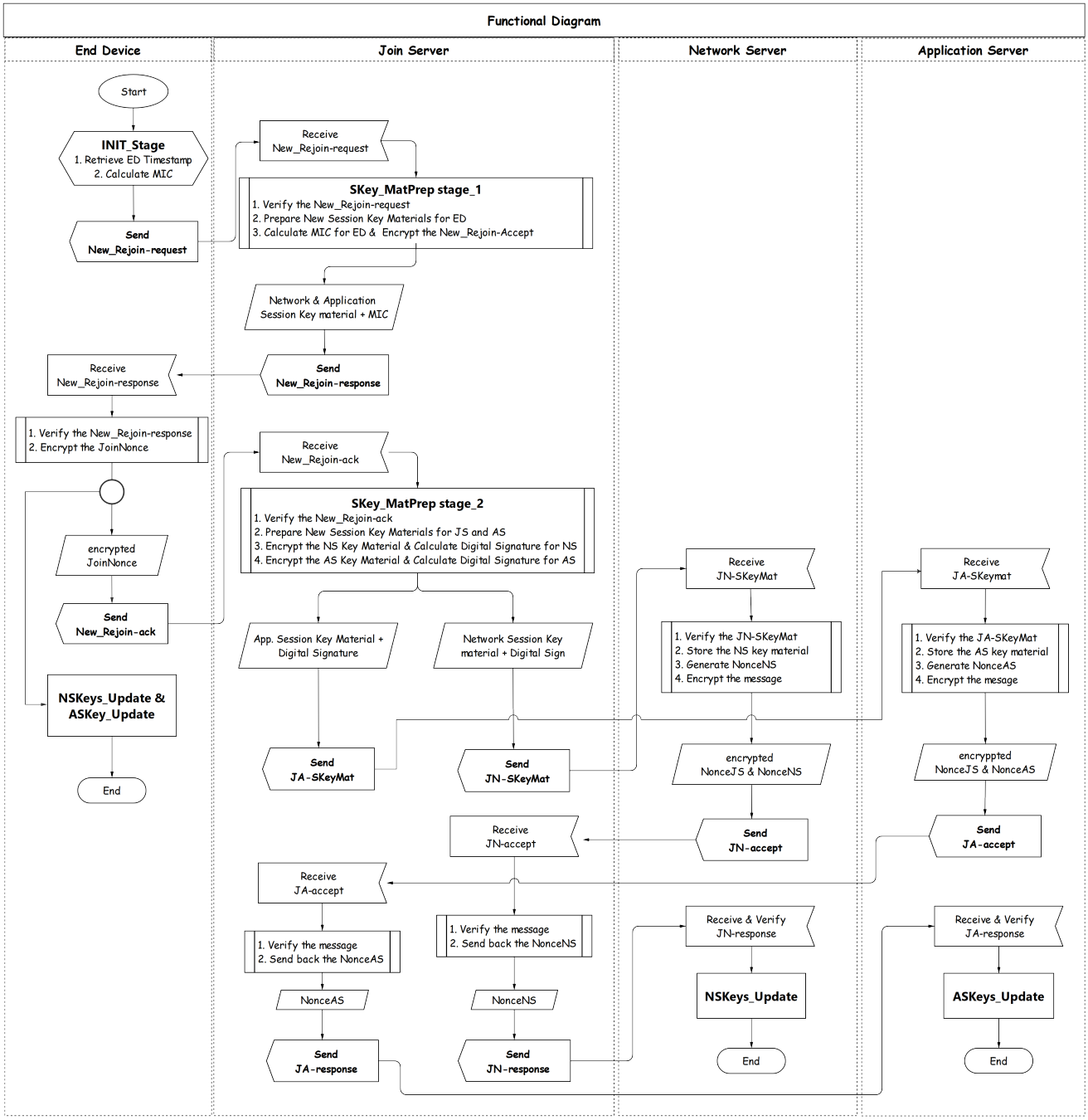

FIGURE 3. The Functional Diagram of the Proposed Session Key Updating Scheme for LoRaWANs.

scheme. The figure shows that the stage starts with the ED, which sends a message to request session keying materials. Then, the JS prepares the keying material by generating and delivering it to the ED, NS, and AS. When the ED, NS, and AS execute the session key update process, the stage ends.

Next, the three stages with their respective steps are explained.

1) INIT_STAGE

The initialization stage occurs in the ED, which has been configured to update the session keys periodically. The configuration should be done during device commissioning. The INIT_Stage consists of three steps as follows:

1. At the scheduled time, the ED retrieves its device timestamp and the LoRaWAN attributes that are required to ask for the new session's keying material. The retrieved data are defined as a *New_Rejoin-message*, which comprises code of $ReJoinType1$, JS identity $JoinEUI$, ED identity $DevEUI$, and increment counter $RJcount1$ with a maximum value of 65,535 [22].
   - $New\_Rejoin\text{-}message = (ReJoinType1||JoinEUI||DevEUI||RJCount1||Ts)$

2. The ED then calculates the MIC value using the $AES\_CMAC128 - bit$ algorithm with the newest *JSIntKey* [16].
   - $cmac_{EJ} = aes128\_cmac$
     $(JSIntKey, MHDR_{ED}||New\_Rejoin\text{-}message)$
   - $MIC_{EJ} = cmac_r$ [0..3]
3. Once the MIC calculation is complete, the ED combines the ED's message header $MHDR_{ED}$ with *New_Rejoin-message* and MIC_{EJ} to form a *New_Rejoin-request* protocol. Then, the ED sends the *New_Rejoin-request* protocol to the JS to request the new sessions' keying material.
   - $New\_Rejoin\text{-}request = \{MHDR_{ED}||New\_Rejoin\text{-}message||MIC_{EJ}\}$

2) SKEY_MATPREP

The JS executes *SKey_MatPrep stage_1* immediately after it receives the *New_Rejoin-request*. Then, the *SKey_MatPrep stage_2* is executed after receives the *New_Rejoin-ack*. The overall *SKey_MatPrep* stages are defined as follows:

1. The substage *Skey_MatPrep stage_1* begins when the JS verifies the received *New_Rejoin-request* by calculating the MIC and deviation timestamp. The deviation timestamp is calculated by subtracting the ED timestamp *Ts* from the JS timestamp Ts'. The JS moves to the next step only when the message passes the verification, i.e., the MIC calculation is correct and $Ts' - Ts \leq \Delta Ts$. In this study, ΔTs must be within the predefined session key update timeframe of the associated ED. When the received *New_Rejoin-request* does not pass verification, the JS halts the step and sends a notification so that the associated ED repeats the process within a specific boundary time.
2. Immediately after completing the verification, the JS prepares the session keying materials to be sent to the associated ED. Two types of session keying materials are prepared, i.e., the network session keying material and application session keying material. In addition to the keying materials, the JS prepares attributes that are used for the default component of the *New_Rejoin-response*, for example, *devaddr*, *DLSettings*, *RxDelay*, and *CFList*. Furthermore, we define the session keying material components that are used in the proposed scheme with the following sequences: The first component of the session keying material comprises master passwords, *MPNet* and *MPApp*, which are each 8 bytes long. Both are pseudorandom bit sequences generated by the RBG module in the JS [40]. the second component of the session keying material is a 1-byte hex number code used to differentiate several session key updates defined based on the LoRaWAN specifications, $0 \times 01 - 0 \times 04$. We assume that the involved entities have stored the code. The third component of the material is a timestamp. The entities that execute the session key update retrieve 4 bytes of their respective devices' timestamps, *Te*, at the scheduled time. The fourth component of the keying material is the server identities, which are directly involved in the key update execution process, namely, the NS and AS identities. In the proposed scheme, we define the AS identity as *AppID* with the same size as *NetID*, 3 bytes. The last component of the keying material is the identity of the associated ED, *DevEUI*, which is 8 bytes long. Thus, the total material size of each session key type is 24 bytes in the following order:
   - $NSKeyMat = MPNet||Code\_0 \times 01/3/4||Te||NetID||DevEUI$
   - $ASKeyMat = MPApp||Code\_0 \times 02||Te||AppID||DevEUI$

   The combination of the keying materials and attributes is described as a *Rejoin-response-message*.
   - $Rejoin\text{-}response\text{-}message = JoinNonce||NetID||DevAddr||DLSettings||RxDelay||CFList||MPNet||MPApp||AppID$
3. When the *Rejoin-response-message* is ready, the JS prepares the security requirement for delivery. The security requirements for delivering the keying materials to the ED are the MIC calculation, encryption, and concatenation with the JS message header. The MIC calculation utilizes aes128cmac with an updatable *JSIntKey* value [16].
   - $cmac_{JE} = aes128cmac(JSIntKey, 0 \times 01||JoinEUI||RJCount1||MHDR_{JS}||Rejoin\text{-}response\text{-}message)$
   - $MIC_{JE} = cmac_{JE}$ [0..3]

   Then, the JS encrypts the *Rejoin-response-message* and MIC_{JE} by employing the AES_ECB 128-bit decryption algorithm, aes128decrypt [22], with the updatable *JSEncKey* value [16]. Furthermore, the JS concatenates the encrypted keying material with the JS message header.
   - $Enc\_Rejoin\text{-}response = aes12decrypt(JSEncKey, Rejoin\text{-}response\text{-}message||MIC_{JE})$
   - $New\_Rejoin\text{-}response = \{MHDR_{JS}, Enc\_Rejoin\text{-}response\}$
4. The last task of *SKey_MatPrep stage_1* is for the JS to send a *New_Rejoin-response* to the ED. *SKey_MatPrep stage_1* stops here, and the JS should wait for an acknowledgment message from the associated ED
   Meanwhile, on the ED side, several processes are executed after receiving the *New_Rejoin-response* message. First, the ED decrypts and verifies the message by calculating MIC_{JE}. When the ED confirms that it has received legitimate new keying materials, the previous keying materials are automatically deleted to optimize memory management. Second, the ED prepares an acknowledgment message. The preparation is

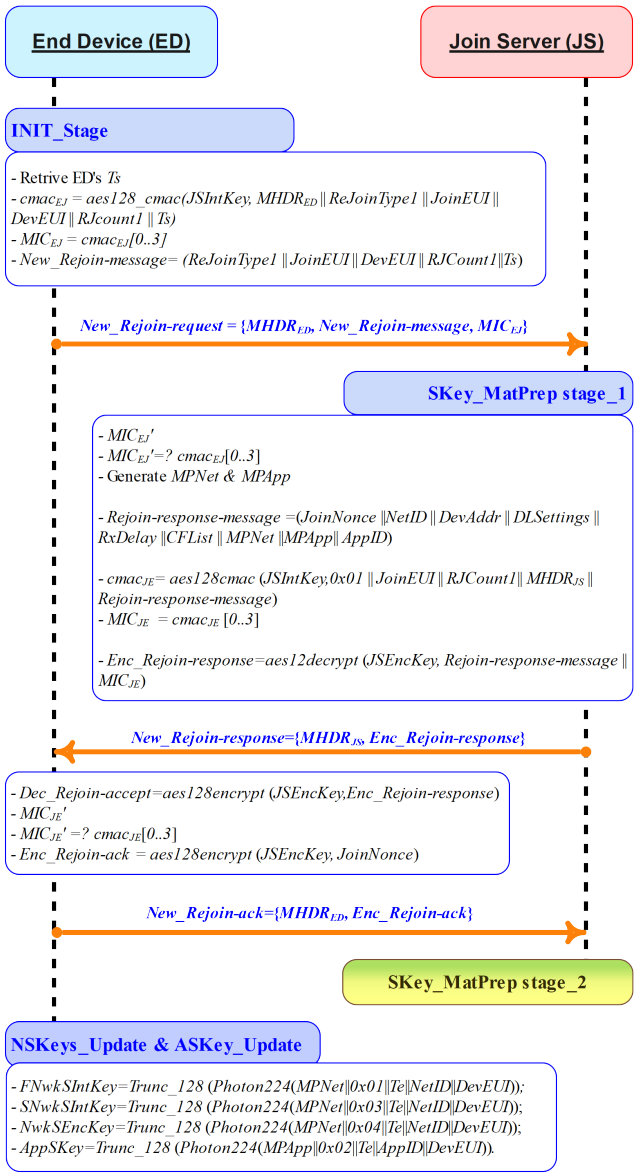

FIGURE 4. The communication protocol exchange between the ED and JS.

performed by encrypting the received *JoinNonce* using AES_ECB 128-bit with the newest *JSEncKey* and then concatenating the result with the ED header $MHDR_{ED}$. Last, the ED sends the message as *New_Rejoin-ack* to the JS. *New_Rejoin-ack* is an acknowledgment message used by the JS as a reference to continue the next substage. The communication protocols exchanged between the ED and JS to request new keying material and deliver the request are illustrated in Fig. 4.

5. The JS directly processes *SKey_MatPrep stage_2* after receiving *New_Rejoint-ack*. The JS decrypts and verifies the message to ensure the ED has received the new session keying material. In *SKey_MatPrep stage_2*, the JS retrieves the session keying material prepared for the related NS and AS and then sets up the security requirements for sending the keying material to the NS and AS separately. The security requirements used for delivering the network session keying materials to the NS are as follows:
   The JS calculates the digital signature with the JS private key.
   - $Sign\_NSKeyMat = ECC256sign(K_{JS}, Hash\,(MPNet\,||DevEUI))$

   The JS combines all the prepared data, including the digest, as *NSKey-message* and encrypts *NSKey-message* using the NS public key to produce *JN-SKeyMat*.
   - $NSKey-message = (JoinEUI||NetID||MPNet||DevEUI||\text{NonceJS}||Sign\_NSKeyMat)$
   - $JN\text{-}SKeyMat = ECC256encrypt\,(P_{NS}, NSKey\text{-}message)$
6. The JS sends *JN-SKeyMat* to the NS.
7. Upon receiving *JN-SKeyMat*, the NS decrypts it using the NS private key K_{NS}.
   - $Dec\_JN\text{-}SKeyMat = ECC256decrypt(K_{NS}, JN\text{-}SKeyMat)$

   Subsequently, the NS calculates and verifies the digest value of the *Sign_NSKeyMat* obtained from the result of *Dec_JN-SKeyMat*. The calculation utilizes the JS public key, P_{JS}:
   - $Hash\text{-}NSKeymat' = Hash(MPNet'|DevEUI')$

   The JS signature is verified if and only if $Hash\text{-}NSKeymat' = Hash(MPNet||DevEUI)$. if it is verified, the NS updates all contexts of the network session key update from the related ED over its individual identity, *DevEUI*.
8. After completing the verification step and updating the database, the NS generates a 1-byte nonce, i.e., *NonceNS*, and secures it by decrypting the value.
   - $Dec\_NonceNS = ECC256decrypt(K_{ns}, NonceNS)$

   The decrypted nonce is then concatenated with *NonceJS* for further encryption. Then, the encryption result becomes the content of the *JN-accept* message. The NS sends *JN-accept* to the associated JS.
   - $JN\text{-}accept = ECC256encrypt(P_{JS}, NonceJS||Dec\_NonceNS)$
9. The JS that receives *JN-accept* performs the reverse steps. It decrypts and then encrypts the message to obtain *NonceNS*. The JS assigns the *NonceNS* value as the content of the *JN-response* and sends the *JN-response* to the NS. The JS uses the *JN-response* to ensure that the NS has acquired the new network session keying materials.
   - $Dec\_JN\text{-}accept = ECC256decrypt(K_{JS}, JN\text{-}accept)$
   - $JN\text{-}response = NonceNS$
10. In the last communication step, the NS obtains $NonceNS'$ through the *JN-response* message. Next, the NS continues its task by comparing $NonceNS'$ with the original *NonceNS* value to validate the received *JN-response* message.
    - $NonceNS' = ?NonceNS$

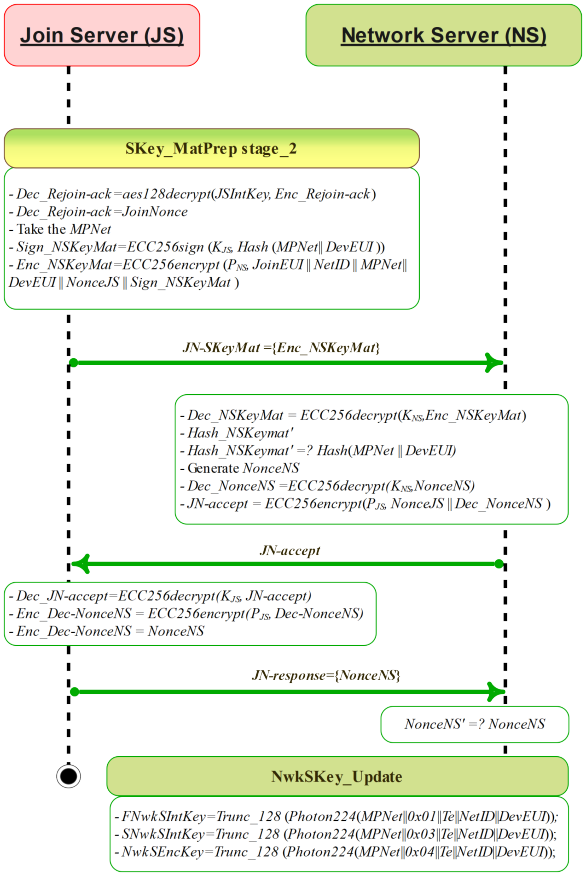

FIGURE 5. The communication protocol exchange between the JS and NS to deliver the network session keying materials.

If the nonce values are equal, then the NS updates the network session key on a regular basis concurrently with other entities, the ED and AS. The communication protocols exchanged between the JS and NS to deliver the network session keying material are illustrated in Fig. 5.

11. The procedure used for transferring application session keying materials from the JS to the AS is similar to the procedure used in JS-NS communication. It starts with steps 5 to 10. However, since the contexts used to update the application session key and network session keys are different, we distinguish the parameters. The distinct parameters are the keying material and its attributes, public-private key usage, nonce value, label, and protocol communication term used to exchange the messages.

3) NSKey_Update and ASKey_Update

NSKey_Update and *ASKey_Update* are the last stages where the session key update algorithm is applied. *NSKey_Update* is a method to update the network session key, and *ASKey_Update* for updating the application session key. The methods apply the *truncated photon* − 225 algorithm. The *photon*−256 algorithm is truncated to the 128 MSBs. The formulas for the *truncatedphoton* − 225 algorithm implemented in *NSKey_Update* and *ASKey_Update* are as follows:

- $FNwkSIntKey = Trunc_{128}$ $(Photon-256\,(MPNet||0\times01||Te||NetID||DevEUI))$;
- $SNwkSIntKey = Trunc_{128}$ $(Photon-256\,(MPNet\,||0\times03\,||Te||NetID||DevEUI))$;
- $NwkSEncKey = Trunc_{128}$ $(Photon-256\,(MPNet\,||0\times04\,||Te||NetID||DevEUI))$;
- $AppSKey = Trunc_{128}$ $(Photon-256\,(MPApp\,||0\times02\,||Te||AppID||DevEUI))$.

IV. RESULTS AND DISCUSSION

This section has four subsections that present the simulation, validation results, and discussion of the proposed scheme. We use the same testing framework as in our previous work [16]. We simulate one of the session key updates and validate the result of the randomness of the new session keys. We provide informal and formal security analyses of the scheme, validate the communication protocol security, and discuss the performance of the proposed scheme. All validations are conducted on the Ubuntu 18.04.5 LTS operating system, which is installed on a virtual box with an 11th Gen Intel(R) Core (TM) i7-11370H processor @ 3.30 GHz, and an 8 GB RAM is used to run the virtualization.

A. VALIDATION OF THE RANDOMNESS OF THE NEW SESSION KEY BIT SEQUENCE

We conduct simulations to validate the randomness of the bit sequences of the new LoRaWAN session keys. The simulations consist of keying material preparation, session key update execution, and randomness validation. Since LoRaWAN has several types of session keys, we select one of the keys as a representative to simulate, i.e., an *FNwkSIntKey*.

The *FNwkSIntKey* requires *NSKeyMat*'s keying material as an input parameter to execute its key update algorithm. Thus, we define *NSKeyMat*'s keying material as shown in Table 3 and generate it using a bash script program. The program generates a unique value of *NSKeyMat* in every iteration. According to Table 3's definitions, the program regenerates *MPNet* in a certain period, while the timestamp varies every nanosecond.

We execute a bash script program for the Photon-256 algorithm to simulate the *FNwkSIntKey* update. The direct output of the Photon-256 simulation is a set of sequences of keys that each have a 224-bit length. Thus, each key sequence is truncated to 128 bits to adjust the LoRaWAN session key length. We define the outcome of the key sequence simulation as a new FNwkSIntKey. The conducted simulation uses 22,650,714 iterations to produce the new FNwkSIntKey, which satisfies the minimum number of bit sequences required for the randomness test, 1000 sequences x 1,000,000 bits [36].

Furthermore, we validate the bit sequence randomness of the new FNwkSIntKey using two standardized statistical test suites, ENT [37], [38], [39] and NIST 800-22 [36], [37]. Table 4 presents the results of the ENT test suite for the entire bit sequence of the new FNwkSIntKey. Table 5 presents the results of the NIST 800-22 test suite for

TABLE 3. The definitions of the input parameters used to simulate FNwkSIntKey key update with the Photon-256 algorithm.

Input parameter	Size (Bytes)	Description
MPNet	8	The master password, MPNet, is changed every seven iterations. If the ED sends payloads daily, then the JS updates the MPNet value once a week.
0x01	1	The 1-byte code used to update the FNwkSIntKey, defined by the LoRaWAN specification. The value remains the same during the simulation.
Te	4	The timestamp value, Te, is always new in each iteration. This represents the fact that the input parameter of the Photon-256 algorithm is always different for every session key.
NetID	3	The value of NetID is never changed throughout the simulation because we assume that the identity of the network server is the same for a single server.
DevEUI	8	A similar situation as above applies to DevEUI, where every end device has a single identity during its lifetime. Thus, during the simulation, DevEUI is set to be static, assuming that the updated session key is only related to a specific ED.

TABLE 4. ENT test results for the new *FNwkSIntKey* bit sequence.

Test No.	Parameter of the ENT Test	Value	Result
1	Entropy	1.000000	Pass
2	Chi squared	0.67	Pass
3	Arithmetic mean	0.5000	Pass
4	Monte Carlo pi	3.141548783	Pass
5	Serial correlation coefficient	0.000002	Pass

the new FNwkSIntKey with the value $\alpha = 0.01$. We use the Fast_NIST_STS_v6.0.1 tools [48], [49] to conduct the statistical test since this method is fast [50].

The validation results prove that the new *FNwkSIntKey* passes all five parameters determined by the ENT statistical test suite and all 15 parameters specified by NIST 800-22. These results indicate that the new session key value has a randomness level that meets the ENT and NIST standard requirements.

B. INFORMAL SECURITY ANALYSIS

This subsection discusses a series of informal security analyses that comprise data confidentiality, integrity protection [18], [19], [20], mutual authentication [20], key secrecy [18], [19], [20], and replay attack resistance [18], [19], [20].

1) DATA CONFIDENTIALITY

The proposed scheme guarantees the confidentiality of session keying material during transmission, but confidentiality during storage on the device is beyond the scope of this research. Another study proposed a hardware security module (HSM) or secure element (SE) scenario to retain the key [51], and this can also be used for the keying material on the device.

The scheme maintains the confidentiality of the keying material data and attributes by implementing two encryption algorithms. The scheme implements an end-to-end

TABLE 5. NIST 800-22 test results for the new *FNwkSIntKey* bit sequence.

Test No.	Parameter of the NIST 800-22 Test	Proportion	Result
1	Frequency	0.9907	Pass
2	BlockFrequency	0.9883	Pass
3	CumulativeSums (1)–(2)	0.9921–0.9879	Pass
4	Runs	0.9893	Pass
5	LongestRun	0.9903	Pass
6	Rank	0.9938	Pass
7	FFT	0.9900	Pass
8	NonOverlapping-Template	0.9852–0.9945	Pass
9	OverlappingTemplate	0.9859	Pass
10	Universal	0.9903	Pass
11	ApproximateEntropy	0.9910	Pass
12	RandomExcursions	0.9831–0.9932	Pass
13	RandomExcursionsVariant	0.9831–0.9915	Pass
14	Serial (1)–(2)	0.9900–0.9928	Pass
15	LinearComplexity	0.9914	Pass

symmetric algorithm, i.e., AES-ECB 128-bit, to encrypt data transmission from the JS to the ED. Later, the scheme also implements the ECC256 encryption algorithm to protect data transmission between servers in JS-NS and JS-AS communications. This is proven by protocol validation utilizing the Scyther tools, as displayed in Fig. 6. The validation results show that the protocol meets the secret test property.

2) INTEGRITY PROTECTION

The integrity protection of the communication protocol in the scheme is achieved by applying the MIC and digital signature. The application of the MIC and digital signature is embedded in the message with the aim of maintaining the integrity of the keying material distributed by the join server. In the scheme, the join server adds the MIC to the message sent to the end device and adds a digital signature to messages sent to other servers, networks and application servers.

At a predetermined time, when the join server receives a request to update the keying material, the join server first verifies the MIC_{EJ} of the request message. Then, the join server prepares new keying material only when the MIC verification is successful. The join server prepares the response message (*New_Rejoin-response*) containing the new keying material with MIC_{JE} and then encrypts and sends the message to the end device. Meanwhile, the end device, as a receiver entity, must verify the MIC_{JE} before utilizing the new keying material.

Additionally, the join server prepares the *NSKey-message* and *ASKey-message;* these messages contain the new keying material embedded with the join server's digital signature. Before sending the messages to the respective servers, the join server encrypts them using each destination server's public key. Thus, only the network and application servers possessing public–private key pairs can verify the signature.

The validation protocol described in Section IV-D proves the statement above. The results of the validation protocol in Fig. 6(a) and (b) show that the protocol has the noninjective synchronization (Nisynch) properties. These properties indicate that the message received by the communicating entity

is precisely the same as the message sent, in both order and content.

3) MUTUAL AUTHENTICATION

The scheme supports mutual authentication among entities to avoid fake join servers and ensure that the intended recipient receives the keying material. The join server authenticates the end device using two properties: the timestamp and MIC_{EJ}, which are embedded through a *New_Rejoin-request* message sent by the end device. The join server verifies the received timestamp and ensures that the ΔTs verification meets the predefined timeframe of the associated ED. Next, the join server verifies the received MIC_{EJ} using a formula that requires an end-to-end symmetric key: *JSIntKey*. The join server trusts the end device only when the request message passes verification.

Meanwhile, the end device authenticates the join server using the MIC_{JE} properties embedded in the *New_Rejoin-response* message. The join server responds to the end device's request by sending the *New_Rejoin-response* message. The end device verifies the MIC_{JE} in the received message to authenticate the join server. Then, to inform the join server when the corresponding end device has completed mutual authentication, the end device sends a *New_Rejoin-ack* message. Immediate exchanged messages between the join server and end device indicate that both sides have successfully authenticated mutually.

Furthermore, the scheme utilizes a digital signature and nonce to support mutual authentication among servers. The network and application servers authenticate the join server using a digital signature. The join server's digital signature embedded in *JN-SKeyMat* and *JA-SKeyMat* is sent to the network and application servers separately. The network and application servers ensure that the join server is a legitimate entity by verifying the received join server's digital signature: *Hash_NSKeyMat* and *Hash_ASKeyMat*.

In contrast, the join server authenticates the network and application servers using a nonce value. The join server only receives the replay of the *JN-SKeyMat* and *JA-SKeyMat* messages when the network and application servers successfully authenticate the join server. In their replay messages, *JN-accept* and *JA-accept*, the network and application servers send back the received *NonceJS* followed by their freshly generated values: *NonceNS* and *NonceAS*. Mutual authentication is achieved when the join server successfully verifies the received nonce value. The join server sends *NonceNS* and *NonceAS* back to each associated server to inform them that the join server has ensured the legitimacy of the network and application servers and confirms that the entities satisfy mutual authentication.

4) PERFECT FORWARD SECRECY

The scheme has a perfect forward secrecy feature for the session key that is achieved through the unpredictable value of its keying material. The keying material has values that are difficult to predict. The keying material consists of five components: two of the five components are a master password that changes periodically and a timestamp that varies every nanosecond. These two dynamic components mean that the keying material is always different in every updating process. The different value applied in each iteration of the truncated Photon-256 algorithm impacts the production of the unique new session keys. The simulation results explained in Section IV-A proved this statement. The perfect forward secrecy of the keying material leads to perfect forward secrecy for the new session key.

An attacker who intends to duplicate the keying material from the end device side should know all the keying material components, including the associated server identity, the timestamp, and the master password. An attacker might know the associated network and application server identities. However, obtaining the other two dynamic keying material components is difficult within the available timeframe. An attacker should know the exact value of the timestamp, which is used only once in every updating scheme. To duplicate the keying material, an attacker must obtain the exact timestamp value. This takes effort because the scheme utilizes a timestamp for up to a nanosecond, which is equivalent to a 32-bit timestamp generated by the end device. Additionally, at the particular time at which an attacker obtains the precise schedule for generating the timestamp and obtains the master password, the gathered data are valid only temporarily. The scheme guarantees perfect forward secrecy since the master password changes regularly in a certain period with an unpredictable value. An attacker who knows the current master password does not know the master password at the previous and later periods. The join server, as the entity that is responsible for updating the master password value, changes its value regularly in every period. In addition, in this scheme, the join server is designed with an RBG module to generate a master password that meets the criteria.

5) REPLAY ATTACK RESISTANCE

The scheme employs timestamps and nonces to resist replay attacks. The end device embeds a timestamp in the request message it sends to the join server. An attacker who attempts to duplicate and then relay the message to the server needs additional time. This condition is unacceptable since the join server checks the ΔTs of the received message. If ΔTs does not match, the related join server drops the message and sends a notification stating that it has detected a replay attack. If an attacker manages to modify the timestamp to comply with the ΔTs requirement, the scheme resists the replay attack by inspecting the MIC_{EJ} calculation. Since the forwarded message has a different timestamp, there will be a MIC_{EJ} mismatch because the scheme sets the timestamp as part of the MIC_{EJ} formula.

In communication between servers, protection against replay attacks is ensured by utilizing the 128-bit nonce value. Before sending out the *JN-SKeyMat* or *JA-SKeyMat* message, the join server encrypts the message along with its digital signature by using each destination server's public key.

TABLE 6. The GNY logical notation.

Notation	Description
$P \lhd X$	P receives X
$P \lhd * X$	P receives X, where X did not originate from P
$P \ni X$	P possess X
$P \mid \equiv X$	P believes X
$P \mid \sim X$	P once said X
$P \mid \equiv \phi(X)$	P recognizes X
$P \mid \equiv \#(X)$	P believes X is fresh
$\{X\}_K$ *and* $\{X\}_K^{-1}$	symmetric encryption and decryption with key K
$P \overset{K}{\leftrightarrow} Q$	K is secretly shared between P and Q
$\{X\}_{+P}$ *and* $\{X\}_{-K}$	encryption with public key +P and private key-K
$P \mid \equiv \overset{+K}{\rightarrow} Q$	P believes +K is the public key for Q
$\frac{P}{Q}$	if P then Q
(X, Y)	X or Y is part of the formula (X, Y)
$F(X)$	a computationally feasible function of X, whose inverse is also computationally feasible

Subsequently, because the fake join server does not have a destination server private key, it cannot decrypt and modify *NonceJS*. Thus, this replay attack is difficult to achieve because the effort required by the fake join server to obtain the *NonceJS* value is equivalent to the effort needed to obtain the asymmetric key pair of the ECC256 algorithm. In a condition in which the fake join server relays *JN-SKeyMa*t or *JA-SKeyMat* messages to the network and application server, the respective server detects this attack since the relayed message has the same *NonceJS* as the one received in the previous message.

C. FORMAL SECURITY ANALYSIS

We perform a formal security analysis of the proposed communication protocols by leveraging Gong, Needham and Yahalom (GNY) logic [52], [53], [54]. The process of formal security analysis is divided into six steps, starting with determining the logical postulates used and ending with proving the protocol's security. To analyze the protocol, GNY provides logical notation, and the GNY logical notation used in this paper is shown in Table 6.

The analysis performed in this section concerns the security of the protocols transmitted between end devices and the join server (ED-JS) and between the join server and the network server (JS-NS). The protocols exchanged between join servers and application servers (JS-AS) have similar steps, with different components being exchanged between the JS and NS. Thus, the formal analysis of JS-AS communication refers to that of JS-NS communication. The details of the steps for proving the security of the session key updating scheme protocol with GNY logic are given below.

1) LOGICAL POSTULATES

The GNY logical postulates employed in this study consist of five types of rules: message interpretation rules $I1$ and $I2$, being-told rules $T1$ and $T4$, possession rules $P1$ and $P2$, recognizability rules $R6$, and freshness rules $F1$. All the rules are presented below.

- $I1 = \frac{P \lhd * \{X\}_K, P \ni K, P \mid \equiv \overset{K}{\rightarrow} Q, P \mid \equiv \phi(X), P \mid \equiv \#(X,K)}{P \mid \equiv Q \mid \sim X, P \mid \equiv Q \mid \sim \{X\}_K, P \mid \equiv Q \ni K}$
- shown at the bottom of the page.
- $T1 = \frac{P \lhd * X}{P \lhd X}$
- $T4 = \frac{P \lhd \{X\}_{+K}, P \ni -K}{P \lhd X}$
- $P1 = \frac{P \lhd X}{P \ni X}$
- $P2 = \frac{P \ni X, P \ni Y}{P \mid \equiv P \ni (X,Y), P \ni F(X,Y)}$
- $R6 = \frac{P \ni H(X)}{P \mid \equiv \phi(X)}$
- $F1 = \frac{P \mid \equiv \#(X)}{P \mid \equiv \#(X,Y), P \mid \equiv \#(F(X))}$

2) PROTOCOL DESCRIPTION

In GNY logic, the protocol description concerns the formalized messages sent and received between entities. In this paper, we interpret the formalized messages of the protocols delivered in ED-JS and JS-NS communication. The following are formalized versions of the protocols exchanged between the ED and JS:

- $\boldsymbol{ED \rightarrow JS}$: $MHDR_{ED}, RejoinType1, JoinEUI, DevEUI, RJCount1, Ts, MIC_{EJ}$ where:
  - $MIC_{EJ} = F\left(\left\{\begin{matrix} MHDR_{ED}, \\ RejoinType1, JoinEUI, \\ DevEUI, RJCount1, Ts \end{matrix}\right\}_{JSIntKey}\right)$
- $\boldsymbol{JS \rightarrow ED}$: $MHDR_{JS}$ $\left\{\begin{matrix} JoinNonce, NetID, \\ Response\text{-}components, \\ MPNet, MPApp, AppID, \\ Ts', MIC_{JE} \end{matrix}\right\}^{-1}_{JSEncKey}$ where:
  - $Response\text{-}components = DevAddr, DLSettings, RxDelay, CFLIst$
  - $MIC_{JE} = F\left(\left\{\begin{matrix} 0 \times 01, JoinEUI, \\ RJCount1, \\ MHDR_{Js}, \\ JoinNonce, NetID, \\ Response\text{-}components, \\ MPNet, MPApp, AppID, \\ Ts' \end{matrix}\right\}_{JSIntKey}\right)$
- $\boldsymbol{ED \rightarrow JS}$: $MHDR_{ED}, \{JoinNonce - 1\}_{JSEncKey}$

Next, the formalized version of the protocols transmitted between the JS and NS is shown below.

- $\boldsymbol{JS \rightarrow NS}$: $\left\{\begin{matrix} \{JoinEUI, NetID, \\ MPNet, DevEUI, \\ NonceJS, \\ Sign\_NSKeyMat \end{matrix}\right\}_{+PNS}$ where:
  - $Sign\_NSKeyMat = \{H(MPNet, DevEUI)\}_{-KJS}$
- $\boldsymbol{NS \rightarrow JS}$: $\{NonceJS - 1, \{NonceNS\}_{-KNS}\}_{+PJS}$
- $\boldsymbol{JS \rightarrow NS}$: $NonceNS - 1$

$$I2 = \frac{P \lhd * \{X < S >\}_{+K}, P \ni (-K, S), P \mid \equiv \overset{+K}{\rightarrow} P, P \mid \equiv P \overset{S}{\leftrightarrow} Q, P \mid \equiv \phi(X, S), P \mid \equiv \#(X, S, +K)}{P \mid \equiv Q \mid \sim (X, < S >), P \mid \equiv Q \mid \sim \{X, < S >\}_{+K}, P \mid \equiv Q \ni +K}$$

3) IDEALIZED MODEL

An idealized model (IM) is used to idealize the formal protocol description given in the previous step. In the ideal form, the response components and all message headers are removed from the protocol. Thus, the idealized model form of the protocol in ED-JS and JS-NS communication based on the GNY logical formulas is as follows:

- $IM1 : JS \triangleleft * (JoinEUI, DevEUI, RJCount1, Ts, MIC_{EJ})$
- $IM2 : ED \triangleleft * \left(\left\{ \begin{array}{c} JoinNonce, \\ NetID, MPNet, \\ MPApp, AppID, \\ Ts', MIC_{JE} \end{array} \right\}_{JSEncKey} \right)$
- $IM3 : JS \triangleleft * \{*JoinNonce\}_{JSEncKey}$
- $IM4 : NS \triangleleft * \left(\left\{ \begin{array}{c} JoinEUI, NetID, \\ MPNet, DevEUI, \\ NonceJS, \\ Sign\_NSKeyMat \} \end{array} \right\}_{+PNS} \right)$
- $IM5 : JS \triangleleft * (\{*NonceJS, \{NonceNS\}_{-KNS}\}_{+PJS})$
- $IM6 : NS \triangleleft * (*NonceNS)$

4) INITIAL ASSUMPTIONS

The initial assumption (IA) expresses the initial data possessed and the belief regarding the data for each entity involved. The initial assumptions stated here are only some of the assumptions applied to prove the defined security goal. Therefore, the initial assumptions stated are only the assumptions of the ED and NS, and the initial assumptions of the JS are omitted because they are not directly applied as a reference in proving the security goals. The initial assumptions used in this analysis are as follows:

- $IA1 : ED \ni JSEncKey$
- $IA2 : ED| \equiv ED \overset{JSEncKey}{\longleftrightarrow} JS$
- $IA3 : ED| \equiv \#(JoinNonce)$
- $IA4 : NS \ni -KNS$
- $IA5 : NS \ni Sign\_NSKeyMat$
- $IA6 : NS| \equiv \overset{+PNS}{\longrightarrow} NS$
- $IA7 : NS| \equiv NS \overset{Sign\_NSKeyMat}{\longleftrightarrow} JS$
- $IA8 : NS| \equiv \#(NonceJS)$

5) SECURITY GOALS

In this analysis, we divide the security goals (SG) into two parts, $SG1$ and $SG2$, to simplify each pair of communication protocol goals. However, generally, the security goal of the proposed protocol is to guarantee that the fresh session's keying material is delivered by trusted entities. The following are the goal sets for the protocol:

- $SG1 : ED| \equiv JS| \sim JoinNonce, NetID, MPNet, MPApp, AppID, Ts', MIC_{JE}$
- $SG2 : NS| \equiv JS| \sim JoinEUI, NetID, MPNet, DevEUI, NonceJS, Sign\_NSKeyMat$

The first security goal means that the ED believes that the JS has sent $JoinNonce, NetID, MPNet, MPApp, AppID, Ts'$, and MIC_{JE}. This message indicates that the JS has successfully verified the end device's request to update the new keying material and replied to it with fresh keying material. The second security goal means that the associated NS believes that the JS has updated its network session keying material by sending freshly generated $JoinEUI, NetID, MPNet, DevEUI, NonceJS, Sign\_NSKeyMat$.

6) SECURITY PROOF

There are two different interpretation postulates applied to prove the security of the protocol as described in step 5. To prove the first goal ($SG1$), that the protocols manage to deliver the new sessions' keying material to the end devices, postulate $I1$ is used. In the use of postulate $I1$, there are specific conditions (C) that must be met, as explained below.

- $C1 : ED \triangleleft * \left\{ \begin{array}{c} JoinNonce, NetID, MPNet, \\ MPApp, AppID, Ts', MIC_{JE} \end{array} \right\}_{JSEncKey}$
- $C2 : ED \ni \text{JSEncKey}$
- $C3 : ED| \equiv \text{ED} \overset{\text{JSEncKey}}{\longleftrightarrow} \text{JS}$
- $C4 : ED| \equiv \phi \left(\begin{array}{c} \text{JoinNonce, NetID, MPNet,} \\ \text{MPApp, AppID, Ts}', \text{MIC}_{\text{JE}} \end{array} \right)$
- $C5 : ED| \equiv \# \left(\begin{array}{c} \text{JoinNonce, NetID, MPNet, MPApp,} \\ \text{AppID, Ts}', \text{MIC}_{\text{JE}}, \text{JSEncKey} \end{array} \right)$

Condition $C1$ is achieved by applying the idealized model $IM2$, condition $C2$ is achieved by applying the initial assumption $A1$, and condition $C3$ is achieved by implementing the initial assumption $A2$. To fulfill condition $C4$, two preconditions are needed, namely,$D1$ and $D2$. The $D1$precondition is achieved by applying the $T1$ postulate to condition $C1$.

- $D1 : ED \triangleleft JoinNonce, NetID, MPNet, MPApp, AppID, Ts', MIC_{JE}$

The second precondition $D2$ is met by applying the postulates of $P1$ to $D1$. The obtained formula of $D2$ is as follows:

- $D2 : ED \ni JoinNonce, NetID, MPNet, MPApp, AppID, Ts', MIC_{JE}$

Next, by applying postulate $R6$ to $D2$, we achieve $C4$. The last condition, $C5$, is achieved by applying postulate $F1$ to initial assumption $IA3$.

The security of the protocol between the end device and the join server is proved by fulfilling all the conditions from $C1$ to $C5$. Therefore, by implementing postulate $I1$ for all five conditions, we obtain the GNY logic formula $ED| \equiv JS| \sim JoinNonce, NetID, MPNet, MPApp, AppID, Ts', MIC_{JE}$. This formula means that the end device believes that the join server has sent a new value of the session keying material. With this proof, it is concluded that the protocol transmitted between the ED and JS meets the logical security requirements according to the objectives set out in the $SG1$ formula.

To prove the second security goal, $SG2$, $I2$ is used. The particular conditions required to apply postulate $I2$ are described as follows:

- $C6 : NS \triangleleft * \left\{ \begin{array}{c} JoinEUI, NetID, MPNet, DevEUI, \\ NonceJS, < Sign\_NSKeyMat > \end{array} \right\}_{+PNS}$
- $C7 : NS \ni (-KNS, Sign\_NSKeyMat)$
- $C8 : NS| \equiv \overset{+PNS}{\longrightarrow} NS$
- $C9 : NS| \equiv NS \overset{Sign\_NSKeyMat}{\longleftrightarrow} JS$
- $C10 : NS| \equiv \phi \left(\begin{array}{c} JoinEUI, NetID, MPNet, DevEUI, \\ NonceJS, Sign\_NSKeyMat \end{array} \right)$

- $C11 : NS| \equiv \#\begin{pmatrix} JoinEUI, NetID, MPNet, DevEUI, \\ NonceJS, Sign\_NSKeyMat, +PNS \end{pmatrix}$

In this analysis, condition $C6$ is achieved by applying the idealized model $IM4$, while condition $C7$ is achieved by applying postulate $P2$ to initial assumptions $IA4$ and $IA5$. $C8$ is fulfilled by applying $IA6$, while $C9$ is obtained by implementing $IA7$. The next two conditions, C10 and C11, can be fulfilled by adding preconditions (D). To obtain $C10$, preconditions $D3$ and $D4$ must be satisfied. By assigning postulate $T4$ to idealized model $IM4$, we obtain $D3$ with the following formula:

- $D3 : NS \triangleleft JoinEUI, NetID, MPNet, DevEUI, NonceJS, Sign\_NSKeyMat$

To satisfy precondition $D4$, we apply postulate $P1$ to D3; then, we obtain D4 as follows:

- $D4 : NS \ni JoinEUI, NetID, MPNet, DevEUI, NonceJS, Sign\_NSKeyMat$

When the two preconditions have been satisfied, condition $C10$ is achieved by applying postulate $R6$ to $D4$. The last condition $C11$ is obtained by implementing the two preconditions $D5$ and $D6$. By applying postulate $F1$ to $IA8$, we obtain $D5$.

- $D5 : NS| \equiv$ #(JoinEUI, NetID, MPNet,DevEUI, NonceJS)

Then, by applying F1 to D5, we obtain D6.

- $D6 : NS| \equiv \#\begin{pmatrix} JoinEUI, NetID, MPNet, DevEUI, \\ NonceJS, Sign\_NSKeyMat \end{pmatrix}$

Furthermore, $C11$ is obtained by utilizing postulate $F1$ on precondition $D6$. Last, the proof of the security of the JS-NS protocol is achieved by applying postulate $I2$ to conditions $C6$ to $C11$. Applying $I2$ to all conditions results in one formula, namely, $NS| \equiv JS| \sim JoinEUI, NetID, MPNet, DevEUI, NonceJS, Sign\_NSKeyMat$. This formula means that the network server believes that the join server has sent freshly generated network session keying material as defined in $SG2$. Based on the above evidence, it is concluded that the protocol transmitted between the join server and the network server meets the logical security requirements.

D. SECURITY VALIDATION OF THE COMMUNICATION PROTOCOLS

We validate the security of the proposed communication protocols using Scyther tools [55], [56], [57]. As explained at the beginning of Section IV, the protocols are validated in an Ubuntu-based environment. We model all protocols transmitted between entities in the form of an $*$.spdl program run on Scyther tools, and Fig. 6 displays the validation results. Fig. 6(a) is the Scyther validation result for the three steps of the protocols exchanged by the ED and JS, and Fig. 6(b) is the result of the communication protocol steps transmitted by the JS and NS in this scheme.

Based on the Scyther verification results in Fig. 6, the protocols in our scheme meet the secrecy property, indicating that the communication protocols between the ED and JS and the JS and NS are secure. Communication secrecy implies that the protocols guarantee the confidentiality of the keying material distribution.

Scyther proves that the protocols have aliveness properties. The aliveness shown in Fig. 6(a) is achieved because the ED includes the timestamp when sending a request message, and the JS only replies to a request when the ΔTs has been successfully verified. Additionally, as shown in Fig. 6(b), JS-NS communication meets the aliveness requirement by exchanging nonce values.

Furthermore, the Scyther results in Fig. 6 (a) and (b) show that the protocols have the noninjective synchronization (Nisynch) property. Nisynch indicates that the messages received by the ED-JS and JS-NS pairs exactly match the messages sent, both in order and in content. This implies that the protocols protect the message integrity.

The noninjective agreement (Niagree) property is also realized in this protocol verification method, which means that both the sender and receiver entities believe that they agree to exchange information.

Finally, the Scyther tools characterize our proposed protocols, and the results show that they have precisely one trace pattern. This pattern characteristic proves that when an entity sends a message, the receiver receives the message directly. This condition means that the two-way communication protocols are secure from active interception.

E. PERFORMANCE ANALYSIS

In the last part of this section, we analyze the proposed scheme's performance in terms of computational cost, communication cost, and storage [54].

1) COMPUTATIONAL COST

In this paper, computational cost (Tc) considers the time used to run a session key update process. This process consists of two sub-processes: retrieve timestamp and execute the algorithm. Thus, to get the computational cost, we calculate the time used to retrieve a new timestamp (Te) and process the key update (Tu).

The calculation of computational cost is done on the server side. We experiment to retrieve 32-bit timestamps and calculate the time it needed. The experiment of timestamp retrieval is conducted with a bash script program with 100,000 iterations. The experiment shows that the average time needed is 2.45 milliseconds. We use this average time as reference data in evaluating the computational cost of timestamp retrieval. Then, to get the Tu data, we count the time required for updating a session key by running one iteration of the Photon-256 algorithm. The test shows that the time needed to execute the session key update algorithm is 0.001 seconds (1 millisecond). Therefore, to calculate the computational cost, we sum up those two experiments' time data. The total time of session key update (Tc) is the sum of the time needed to retrieve a new timestamp (Te) and to execute the updating algorithm (Tu), $Tc = Te+Tu$. According to the experiment, the computational cost requires 3.45 milliseconds, which is a very short time

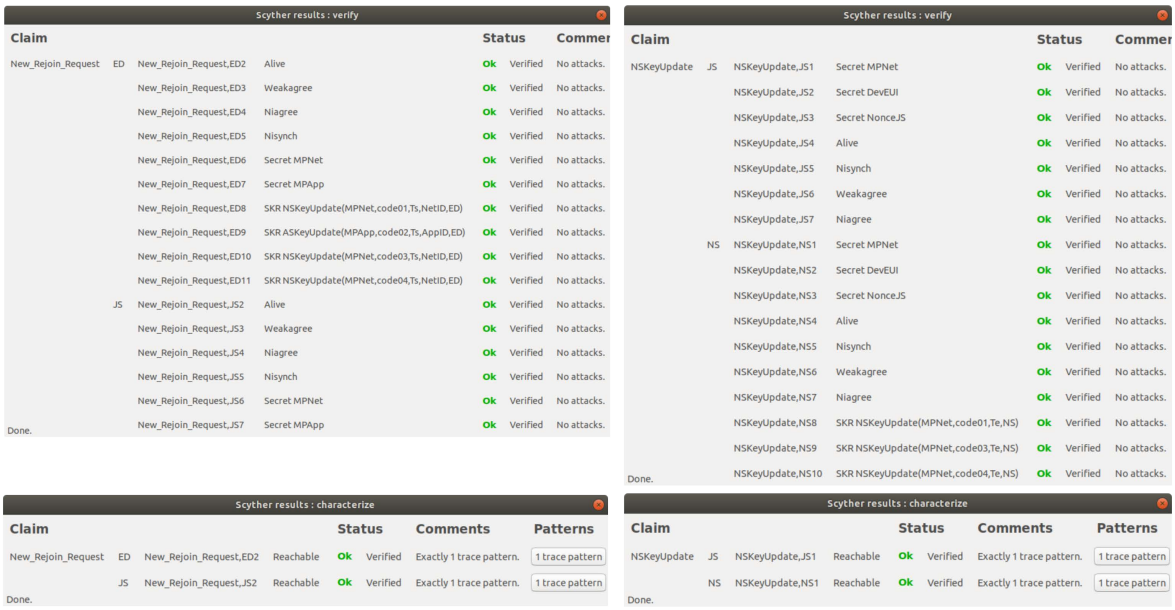

FIGURE 6. Scyther validation result: verification and characterization of the protocols exchanged (a) between the ED and JS and (b) between the JS and NS.

that is required to perform such processing; hence, the computational cost is considered to add no burden to the backend system side.

2) COMMUNICATION COST

In this study, the communication cost refers to the protocol steps sent and received by a pair of entities. These communication costs represent the amount of data traffic transmitted between parties. It should be noted that in wireless data transmission, the heavier the traffic is, the greater the potential for data collision. In the event of a collision, the party needs to process data retransmission. Consequently, if retransmission occurs on the end device, this communication cost incurs burdens for the end device. Therefore, in this section, we analyze the communication cost between the end device and server.

Previous studies assigned the server to update the session key value; Xia *et al.* assigned the KDS [19], and Tsai *et al.* assigned the network server [18], [20]. Thus, when every communication session implements a different key, the previous solutions lead to heavy communication traffic between the assigned server and end device entities because the server has to distribute new key values. This situation does not occur in our scheme because the server sends the keying material component, which is part of the input parameter in updating the session key, instead of directly sending the new key value. In our proposed scheme, traffic is minimized because the keying material component transmission occurs only at a particular periodic time. In situations where a session key is created for each data transmission, our solution incurs less communication cost for the end device than the existing solutions.

Furthermore, assuming that every protocol step costs time T, each pair of an end device and join server needs 3T to update the keying material values. This 3T cost equals the cost required to update the session keys in previous works [19], [18], and [20] between their end devices and assigned servers. Assuming that the end device always uses a new session key to send the data each day, in a week, it needs seven new session keys. When our scheme is set up to update the keying material once a week, the communication cost incurred is 3T. If this scenario is applied to the methods of the previous studies, then the communication cost for each of their solutions is 21T. Thus, our scheme communication cost requires less traffic than the previous solutions, yet it offers more robust security for LoRaWAN by producing a new key in every communication session. A comparison of the communication costs among the solutions for the above scenario is displayed in Table 7.

TABLE 7. Comparison of communication costs.

Scenario: LoRaWAN working for seven days and sending one set of data every day.	Number of end devices	Scheme			
		Tsai et al. [18]	Xia et al. [19]	Tsai et al. [20]	Proposed Scheme
Number of key updates in seven days (Times)	1	7	7	7	7
	N	7N	7N	7N	7N
Communication cost (T) needed to update the key between the end device and the server	1	21T	21T	21T	3T
	N	21NT	21NT	21NT	3NT

3) STORAGE

Additional storage is a concern for end devices since they tend to have a small memory capacity. Thus, we calculate

TABLE 8. Summary and comparison of the proposed scheme.

LoRaWAN session key update scheme	Tsai et al. [18]	Xia et al. [19]	Tsai et al. [20]	Proposed Scheme
Session key update algorithm	SeLPC: Simplified AES	One-way hash function	LPADA: AES	Truncated Photon-256
Communication protocol to update the session key	NS-ED	KDS-ED	*EN-JS	ED-JS
	NS-AS	KDS-NS	JS-NS	JS-NS
		KDS-AS	JS-AS	JS-AS
Data distributed by the server	new session key	new session key	new session key	new session keying material components
Randomness test of the new session key bit sequence	NA	NA	NA	NIST 800-22 and ENT Statistical Test Suites
Data confidentiality	NA	√	√	√
Integrity protection	√	√	√	√
Mutual authentication	NA	NA	√	√
Perfect forward secrecy	√	√	√	√
Replay attack resistance	√	√	√	√
Formal security analysis of the protocols	NA	NA	NA	Formally verified by leveraging GNY logic
Protocol security validation	NA	NA	NA	Proven secure by using Scyther tools
**Communication cost	21NT	21NT	21NT	3NT
Extra storage for the end device	NA	NA	NA	19 bytes

NA: Not available
√ : Achieved
*EN: End Node as defined by Tsai et al. [20]
**Communication cost in the described scenario (N: number of end devices, T: communication cost)

the extra keying material components and ignore the default components owned by the end device. The total amount of extra storage required by the scheme is only 19 bytes, specifically, 8 bytes for *MPNe*t, 8 bytes for *MPApp*, and 3 bytes for *AppID*. Subsequently, to optimize memory management, the end device automatically deletes the previous keying material components once it confirms that it has received legitimate new keying material components. Then, compared to all storage on the end device [58], the additional storage for the keying material components is considered negligible. Thus, there is an insignificant additional storage impact of this solution to significantly enhance LoRaWAN security with a more secure communication session.

A summary of our proposed scheme and comparison to previous studies are displayed in Table 8.

V. CONCLUSION

This study proposed a novel session key update scheme to enhance LoRaWAN security by providing different keys in every communication session. The scheme consists of three stages, and these stages use a truncated Photon-256 algorithm as well as a set of secure and cost-effective communication protocols. The bit sequences of the keys produced by the algorithm pass the NIST 800-22 and ENT statistical test suites. Additionally, the communication protocols ensure the confidentiality and integrity of the data, mutual authentication, and perfect forward secrecy, and they resist replay attacks. The security of the protocols is formally verified by GNY logic and validated by the Scyther tools. The performance analysis shows that the scheme is more cost effective than those of previous studies. A limitation of this study is that the proposed scheme was not fully implemented in the LoRaWAN architecture. Consequently, future research will investigate this limitation.

REFERENCES

[1] S. Fewkes. (Dec. 7, 2021). *LoRaWAN? Formally Recognized as ITU International Standard for Low Power Wide Area Networking*. Fremont, CA USA. Accessed: Dec. 9, 2021. [Online]. Available: https://lora-alliance.org/lora-alliance-press-release/lorawan-formally-recognized-as-itu-international-standard-for-low-power-wide-area-networking/

[2] International Telecommunication Union (ITU). (Dec. 1, 2021). *Y.4480: Low Power Protocol for Wide Area Wireless Networks*. Accessed: Mar. 19, 2022. [Online]. Available: https://www.itu.int/t/aap/recdetails/10096

[3] J. Blackman. (Dec. 8, 2021). *LoRaWAN Gets the Nod From the ITU as Proper IoT Standard*. Accessed: Feb. 19, 2022. [Online]. Available: https://enterpriseiotinsights.com/20211208/channels/news/lorawan-gets-the-nod-from-the-itu-as-proper-iot-standard

[4] Semtech. (2022). *LoRa Technology Is Connecting Our Smart Planet*. [Online]. Available: https://www.semtech.com/lora/lora-applications

[5] LoRa Alliance. *Smart Industry*. Accessed: Feb. 25, 2022. [Online]. Available: https://lora-alliance.org/industry-vertical-market/ and https://lora-alliance.org/lorawan-vertical-markets/industry

[6] E. Sisinni, D. F. Carvalho, P. Ferrari, A. Flammini, D. R. C. Silva, and I. M. D. Da Silva, "Enhanced flexible LoRaWAN node for industrial IoT," in *Proc. 14th IEEE Int. Workshop Factory Commun. Syst. (WFCS)*, Imperia, Italy, Jun. 2018, pp. 1–4, doi: 10.1109/WFCS.2018.8402367.

[7] A. Vitadhani, F. Alief, B. Haryanto, R. Harwahyu, and R. F. Sari, "Simulating LoRaWAN for flood early warning system in Ciliwung River, Bogor-Jakarta," in *Proc. Int. Seminar Appl. Technol. Inf. Commun. (iSemantic)*, Semarang, Indonesia, Sep. 2020, pp. 274–279, doi: 10.1109/iSemantic50169.2020.9234221.

[8] A. D. Prajanti, B. Wahyuaji, F. B. Rukmana, R. Harwahyu, and R. F. Sari, "Performance analysis of LoRa WAnTechnology for optimum deployment of Jakarta smart city," in *Proc. 2nd Int. Conf. Informat. Comput. Sci. (ICICoS)*, Semarang, Indonesia, Oct. 2018, pp. 1–6, doi: 10.1109/ICICOS.2018.8621803.

[9] S. Sugianto, A. A. Anhar, R. Harwahyu, and R. F. Sari, "Simulation of mobile LoRa gateway for smart electricity meter," in *Proc. 5th Int. Conf. Electr. Eng., Comput. Sci. Informat. (EECSI)*, Malang, Indonesia, Oct. 2018, pp. 292–297, doi: 10.1109/EECSI.2018.8752818.

[10] A. T. Nugraha, N. Hayati, and M. Suryanegara, "The experimental trial of LoRa system for tracking and monitoring patient with mental disorder," in *Proc. Int. Conf. Signals Syst. (ICSigSys)*, Bali, Indonesia, May 2018, pp. 191–196, doi: 10.1109/ICSIGSYS.2018.8372663.

[11] N. Hayati and M. Suryanegara, "The IoT LoRa system design for tracking and monitoring patient with mental disorder," in *Proc. IEEE Int. Conf. Commun., Netw. Satell. (Comnetsat)*, Semarang, Indonesia, Oct. 2017, pp. 135–139, doi: 10.1109/COMNETSAT.2017.8263587.

[12] A. T. Nugraha, R. Wibowo, M. Suryanegara, and N. Hayati, "An IoT-LoRa system for tracking a patient with a mental disorder: Correlation between battery capacity and speed of movement," in *Proc. 7th Int. Conf. Comput. Commun. Eng. (ICCCE)*, Kuala Lumpur, Malaysia, Sep. 2018, pp. 198–201, doi: 10.1109/ICCCE.2018.8539316.

[13] T. C. M. Donmez and E. Nigussie, "Key management through delegation for LoRaWAN based healthcare monitoring systems," in *Proc. 13th Int. Symp. Med. Inf. Commun. Technol. (ISMICT)*, Oslo, Norway, May 2019, pp. 1–6, doi: 10.1109/ISMICT.2019.8743947.

[14] J. Han and J. Wang, "An enhanced key management scheme for LoRaWAN," *Cryptography*, vol. 2, no. 4, p. 34, Nov. 2018, doi: 10.3390/cryptography2040034.

[15] J. Xing, L. Hou, K. Zhang, and K. Zheng, "An improved secure key management scheme for LoRa system," in *Proc. IEEE 19th Int. Conf. Commun. Technol. (ICCT)*, Xi'an, China, Oct. 2019, pp. 296–301, doi: 10.1109/ICCT46805.2019.8947215.

[16] N. Hayati, K. Ramli, S. Windarta, and M. Suryanegara, "A novel secure root key updating scheme for LoRaWANs based on CTR_AES DRBG 128," *IEEE Access*, vol. 10, pp. 18807–18819, 2022, doi: 10.1109/ACCESS.2022.3150281.

[17] X. Chen, M. Lech, and L. Wang, "A complete key management scheme for LoRaWAN v1.1," *Sensors*, vol. 21, no. 9, p. 2962, Apr. 2021.

[18] K.-L. Tsai, Y.-L. Huang, F.-Y. Leu, I. You, Y.-L. Huang, and C.-H. Tsai, "AES-128 based secure low power communication for LoRaWAN IoT environments," *IEEE Access*, vol. 6, pp. 45325–45334, 2018, doi: 10.1109/ACCESS.2018.2852563.

[19] Z. Xia, H. Zhou, K. Gu, B. Yin, Y. Zeng, and M. Xu, "Secure session key management scheme for meter-reading system based on LoRa technology," *IEEE Access*, vol. 6, pp. 75015–75024, 2018, doi: 10.1109/ACCESS.2018.2883657.

[20] K.-L. Tsai, F.-Y. Leu, I. You, S.-W. Chang, S.-J. Hu, and H. Park, "Low-power AES data encryption architecture for a LoRaWAN," *IEEE Access*, vol. 7, pp. 146348–146357, 2019, doi: 10.1109/ACCESS.2019.2941972.

[21] *LoRaWANTM Security Frequently Asked Questions*. Accessed: Dec. 23, 2021. [Online]. Available: https://lora-alliance.org/wp-content/uploads/2020/11/la_faq_security_0220_v1.2_0.pdf

[22] LoRa Alliance Technical Committee. (Oct. 11, 2017). *LoRaWANTM 1.1 Specification*. LoRa Alliance. Accessed: Sep. 25, 2019. [Online]. Available: https://lora-alliance.org/resource_hub/lorawan-specification-v1-1/

[23] E. Barker, "Recommendation for key management part 1: General," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. NIST SP 800-57pt1r4, Jan. 2016, doi: 10.6028/NIST.SP.800-57pt1r4.

[24] F. E. Heath. (May 2017). *LPWA Technology Security Comparison A White Paper From Franklin Heath Ltd*. Accessed: Feb. 17, 2021. [Online]. Available: https://www.semanticscholar.org/paper/LPWA-Technology-Security-Comparison-A-White-Paper-Heath/f5b1d1907ab3f64cee150dbba3b709a29f3c906f

[25] Gemalto, Actility, and Semtech. *LoRaWANTM Security A White Paper*. Accessed: Dec. 3, 2021. [Online]. Available: https://lora-alliance.org/sites/lorawan_security_whitepaper.pdf

[26] The Things Industries. *Activate Your Device*. [Online]. Available: https://www.thethingsnetwork.org/docs/devices/uno/quick-start.html#get-your-device-eui

[27] N. SORNIN (Semtech). (Oct. 11, 2017). *LoRaWANTM Backend Interfaces 1.0 3 Specification*. LoRa AllianceTM. Accessed: Sep. 25, 2019. [Online]. Available: https://lora-alliance.org/resource-hub/lorawanr-back-end-interfaces-v10

[28] N. Sornin, M. Luis, T. Eirich, T. Kramp, and O. Hersent. (Jan. 2015). *LoRaWAN Specification V1.0*. LoRa Alliance. Accessed: Apr. 25, 2021. [Online]. Available: https://lora-alliance.org/resource_hub/lorawan-specification-v1-0/

[29] N. Sornin. (Feb. 2016). *LoRaWAN Specification Version: V1.0.1*. LoRa Alliance. Accessed: May 5, 2021. [Online]. Available: https://lora-alliance.org/resource_hub/lorawan-specication-v1-0-1/

[30] N. Sornin, M. Luis, T. Eirich, T. Kramp, and O. Hersent. (Jul. 2016). *LoRaWAN Specification Version: V1.0.2*. LoRa Alliance. Accessed: May 5, 2021. [Online]. Available: https://lora-alliance.org/resource_hub/lorawan-specification-v1-0-2/

[31] N. Sornin. (Jul. 2018). *LoRaWANTM 1.0.3 Specification*. LoRa Alliance. Accessed: May 5, 2021. [Online]. Available: https://loraalliance.org/resource_hub/lorawan-specification-v1-0-3/

[32] E. Barker and W. C. Barker, "Recommendation for key management: Part 2–best practices for key management organizations," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. NIST SP 800-57pt2r1, May 2019, doi: 10.6028/NIST.SP.800-57pt2r1.

[33] S. A. A. Hakeem, S. M. A. El-Kader, and H. Kim, "A key management protocol based on the hash chain key generation for securing LoRaWAN networks," *Sensors*, vol. 21, no. 17, p. 5838, Aug. 2021, doi: 10.3390/s21175838.

[34] 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, doi: 10.1109/ACCESS.2020.2978100.

[35] D. Giry. (May 24, 2020). *Cryptographic Key Length Recommendation*. Accessed: Dec. 20, 2021. [Online]. Available: https://www.keylength.com

[36] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and E. Barker, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," Booz Allen Hamilton, McLean, VA, USA, 2001, p. 131.

[37] S. Kalanadhabhatta, D. Kumar, K. K. Anumandla, S. A. Reddy, and A. Acharyya, "PUF-based secure chaotic random number generator design methodology," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 7, pp. 1740–1744, Jul. 2020, doi: 10.1109/TVLSI.2020.2979269.

[38] J. Walker. (2008). *A Pseudorandom Number Sequence Test Program*. Accessed: Oct. 2, 2022. [Online]. Available: https://www.fourmilab.ch/random/

[39] C. Camara, H. Martin, P. Peris-Lopez, and L. Entrena, "A true random number generator based on gait data for the Internet of you," *IEEE Access*, vol. 8, pp. 71642–71651, 2020, doi: 10.1109/ACCESS.2020.2986822.

[40] National Institute of Standards and Technology, "Security requirements for cryptographic modules," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. NIST FIPS 140-3, Apr. 2019, doi: 10.6028/NIST.FIPS.140-3.

[41] I. Muchtadi-Alamsyah and Y. B. W. Tama, "Implementation of elliptic curve25519 in cryptography," in *Theorizing STEM Education in the 21st Century*, K. G. Fomunyam, Ed. Rijeka, Croatia: IntechOpen, 2020, doi: 10.5772/intechopen.88614.

[42] D. J. Bernstein, "Curve25519: New Diffie-Hellman speed records," in *Public Key Cryptography—PKC*, vol. 3958, M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds. Berlin, Germany: Springer, 2006, pp. 207–228, doi: 10.1007/11745853_14.

[43] C. A. Lara-Nino, A. Diaz-Perez, and M. Morales-Sandoval, "Elliptic curve lightweight cryptography: A survey," *IEEE Access*, vol. 6, pp. 72514–72550, 2018, doi: 10.1109/ACCESS.2018.2881444.

[44] N. Hayati, K. Ramli, M. Suryanegara, and Y. Suryanto, "Potential development of AES 128-bit key generation for LoRaWAN security," in *Proc. 2nd Int. Conf. Commun. Eng. Technol. (ICCET)*, Nagoya, Japan, Apr. 2019, pp. 57–61, doi: 10.1109/ICCET.2019.8726884.

[45] P. M. Mukundan, S. Manayankath, C. Srinivasan, and M. Sethumadhavan, "Hash-one: A lightweight cryptographic hash function," *IET Inf. Secur.*, vol. 10, no. 5, pp. 225–231, Sep. 2016, doi: 10.1049/iet-ifs.2015.0385.

[46] D.-H. Bui, "An innovative lightweight cryptography system for Internet-of-Things ULP applications," Ph.D. dissertation, Vietnam, Hanoi, 2019, p. 137.

[47] *Information Technology—Security Techniques—Lightweight Cryptography—Part 5: Hash-Functions*, Standard ISO/IEC 29192-5:2016, International Organization for Standardization, Aug. 2016. Accessed: Dec. 29, 2021. [Online]. Available: https://www.iso.org/standard/67173.html

[48] Z. Říha and M. Sýs. *Fast_NIST_STS_v6.0.1*. Accessed: Dec. 1, 2021. [Online]. Available: https://randomness-tests.fi.muni.cz

[49] M. Sýs and Z. Říha, "Faster randomness testing with the NIST statistical test suite," in *Security, Privacy, and Applied Cryptography Engineering*, vol. 8804, R. S. Chakraborty, V. Matyas, and P. Schaumont, Eds. Cham, Switzerland: Springer, 2014, pp. 272–284, doi: 10.1007/978-3-319-12060-7_18.

[50] M. Sýs, Z. Říha, and V. Matyáš, "Algorithm 970: Optimizing the NIST statistical test suite and the Berlekamp-Massey algorithm," *ACM Trans. Math. Softw.*, vol. 43, no. 3, pp. 1–11, Jan. 2017, doi: 10.1145/2988228.

[51] Y. Jeon, H.-I. Ju, and S. Yoon, "Design of an LPWAN communication module based on secure element for smart parking application," in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, Las Vegas, NV, USA, Jan. 2018, pp. 1–2, doi: 10.1109/ICCE.2018.8326112.

[52] L. Gong, R. Needham, and R. Yahalom, "Reasoning about belief in cryptographic protocols," in *Proc. IEEE Comput. Soc. Symp. Res. Secur. Privacy*, Oakland, CA, USA, 1990, pp. 234–248, doi: 10.1109/RISP.1990.63854.

[53] F. Zhu, "SecMAP: A secure RFID mutual authentication protocol for healthcare systems," *IEEE Access*, vol. 8, pp. 192192–192205, 2020, doi: 10.1109/ACCESS.2020.3032541.
[54] M. Sidorov, M. T. Ong, R. V. Sridharan, J. Nakamura, R. Ohmura, and J. H. Khor, "Ultralightweight mutual authentication RFID protocol for blockchain enabled supply chains," *IEEE Access*, vol. 7, pp. 7273–7285, 2019, doi: 10.1109/ACCESS.2018.2890389.
[55] C. J. F. Cremers, "The Scyther tool: Verification, falsification, and analysis of security protocols," in *Computer Aided Verification*, vol. 5123, A. Gupta and S. Malik, Eds. Berlin, Germany: Springer, 2008, pp. 414–418, doi: 10.1007/978-3-540-70545-1_38.
[56] C. Cremers and S. Mauw, *Operational Semantics and Verification of Security Protocols*. Berlin, Germany: Springer, 2012, doi: 10.1007/978-3-540-78636-8.
[57] F. A. Putra, K. Ramli, N. Hayati, and T. S. Gunawan, "PURA-SCIS protocol: A novel solution for cloud-based information sharing protection for sectoral organizations," *Symmetry*, vol. 13, no. 12, p. 2347, Dec. 2021, doi: 10.3390/sym13122347.
[58] STMicroelectronics. (2022). *LoRaWAN Products*. Accessed: Jun. 1, 2022. [Online]. Available: https://www.st.com/en/wireless-connectivity/lorawan-products.html#overview

NUR HAYATI (Member, IEEE) received the bachelor's degree in applied science majoring in telecommunication engineering from Politeknik Elektronika Negeri Surabaya, in 2010, and the master's degree in computer engineering from Universitas Indonesia, in 2015. She is currently pursuing the Ph.D. degree with Universitas Indonesia. She is an Electrical Engineering Lecturer for undergraduate students with the Faculty of Engineering, Universitas Muhammadiyah Yogyakarta. Her research interests include computer networks, the IoT security, and security protocol.

SUSILA WINDARTA (Member, IEEE) received the Diploma degree in cryptography from the National Crypto Academy, Bogor, Indonesia, the bachelor's degree in information systems from Gunadarma University, Indonesia, and the master's degree in mathematics from the Department of Mathematics, Faculty of Mathematics and Natural Sciences, Universitas Indonesia, Depok, Indonesia. He is currently pursuing the doctoral degree with the Department of Electrical Engineering, Faculty of Engineering, Universitas Indonesia. Since 2013, he has been working as a Lecturer with the Department of Cyber-Security Engineering, National Cyber and Crypto Polytechnic, Indonesia. His research interests include cryptography and information security-related topics, especially cryptographic hash function and security protocol.

MUHAMMAD SURYANEGARA (Senior Member, IEEE) received the bachelor's degree in electrical engineering from Universitas Indonesia, in 2003, the master's degree from the University College, London, U.K., in 2004, and the Ph.D. degree from the Tokyo Institute of Technology, Japan, in 2011. He is an Associate Professor with Telecommunications Management, Universitas Indonesia. Currently, he is a Lecturer with the Graduate Program in Telecommunications Management, Department of Electrical Engineering, Universitas Indonesia. He has published more than 75 academic articles as a main author or coauthor. He is the Principal Investigator on the research area of ICT policy and technology management, the IoT, 4G/5G and wireless communication technology, and concerning both technical research and regulatory aspects. He engaged in international regulatory activities, among others as the Drafting Group Chairman of the Asia Pacific Telecommunity (APT) Preparatory Group for ITU's World Radiocommunications Conference (WRC-15) and WRC-19.

BERNARDI PRANGGONO (Senior Member, IEEE) received the B.Eng. degree in electronics and telecommunication engineering from Waseda University, Japan, the M.DigComms. degree in digital communications from Monash University, Australia, and the Ph.D. degree in electronics and electrical engineering from the University of Leeds, U.K. He has held academic and research positions at the Glasgow Caledonian University, Queen's University Belfast, and the University of Leeds. He has held industrial positions at Oracle, PricewaterhouseCoopers, Accenture, and Telstra. He is currently a Senior Lecturer with the Department of Engineering and Mathematics, Sheffield Hallam University. His current research interests include cybersecurity, the Internet of Things, cloud computing, and green ICT. He is an Associate Editor of *Frontiers of Computer Science* and *Frontiers in Communications and Networks*. He is a fellow of the Higher Education Academy (HEA).

KALAMULLAH RAMLI (Member, IEEE) received the master's degree in telecommunication engineering from the University of Wollongong, NSW, Australia, in 1997, and the Ph.D. degree in computer networks from Universitaet Duisburg-Essen (UDE), NRW, Germany, in 2000, and the doctoral degree, in 2003. He currently teaches advanced communication networks, embedded systems, object-oriented programming, and engineering and entrepreneurship. He has been a Lecturer with Universitas Indonesia (UI), since 1994, and a Professor of computer engineering, since 2009. He is a prolific author, having published over 125 journal/conference papers and having written eight books/book chapters. His research interests include embedded systems, information and data security, computers and communications, and biomedical engineering.