Journal of Sensor and Actuator Networks

MDPI

Article

A Trusted Security Key Management Server in LoRaWAN: Modelling and Analysis

Koketso Ntshabele [1], Bassey Isong [1,*], Naison Gasela [1] and Adnan M. Abu-Mahfouz [2]

[1] Computer Science Department, North-West University, Mafikeng 2745, South Africa
[2] Council of Scientific and Industrial Research (CSIR), Pretoria 0001, South Africa
* Correspondence: bassey.isong@nwu.ac.za

Abstract: The traditional Long-Range Wide-Area Network (LoRaWAN) uses an Advanced Encryption Standard (AES) 128 bit symmetric key to secure entities and data against several attacks. However, due to the existence of heterogeneous applications, designing a globally accepted and resilient LoRaWAN security model is challenging. Although several security models to maximize the security efficiency in LoRaWAN exist using the trusted key server to securely manage the keys, designing an optimum LoRaWAN security model is yet to be fully realized. Therefore, in this paper, we proposed two LoRaWAN security algorithms, A and B, for a trusted key management server (TKMS) to securely manage and distribute the keys amongst the entities. Algorithm B is an enhanced version of Algorithm A, which utilizes the security shortcomings of Algorithm A. We employed two formal analysis methods in the modelling, results analysis, and verification. The Scyther security verification tool was used for algorithm modelling and analysis against all possible attacks, while BAN logic was used to prove the logical correctness of the proposed algorithms. The results indicate that BAN logic feasibly proves the model logic correctness and the security claims employed in Scyther are reliable metrics for assessing the algorithms' security efficiency. The security claims proved that the security algorithm is more secure and reliable as no attacks were detected across all entities in the enhanced-Algorithm B, unlike in Algorithm A. Moreover, the application of hashing minimizes computation cost and time for authentication and message integrity as compared to symmetric and asymmetric encryption. However, the proposed algorithm is yet to be verified as completely lightweight.

Keywords: IoT; LoRa; LoRaWAN; attacks; key security; security model; symmetric encryption

Citation: Ntshabele, K.; Isong, B.; Gasela, N.; Abu-Mahfouz, A.M. A Trusted Security Key Management Server in LoRaWAN: Modelling and Analysis. *J. Sens. Actuator Netw.* **2022**, *11*, 52. https://doi.org/10.3390/jsan11030052

Academic Editors: Mohamed Benbouzid, Leandros Maglaras and Mohamed Amine Ferrag

Received: 11 August 2022
Accepted: 25 August 2022
Published: 5 September 2022

1. Introduction

The Internet of Things (IoT) is a promising wireless communication platform that allows several devices to autonomously share data and communicate over the Internet. This technology is widely applied in several areas such as in the production industries, agriculture, healthcare, transportation, and homes. However, IoT devices are resource constrained, which limits their applicability. Thus, to improve and extend IoT's flexibility, the Long-Range Wide-Area Network (LoRaWAN) was introduced for IoT-based applications. LoRaWAN is a popular Medium Access Control (MAC) protocol that has several benefits such as wide-area coverage, long-range communication, low deployment cost, and low energy consumption [1–12]. LoRaWAN is deployed on top of the Long-Range (LoRa) protocol, which is a physical layer protocol. Interoperability and cross-cooperation can be experienced by the communicating entities in IoT-based LoRaWAN applications with less complex implementations; however, this implementation raises a lot of security and privacy concerns as part of the transmitted data can be sensitive information [1–12].

Security model design in LoRaWAN is dominated by cryptographic techniques such as Advanced Encryption Standard (AES) 128 bit symmetric encryption, which is used to

generate encryption keys to secure the transmitted data between the entities. This is important to increase the strength of the security models being deployed in IoT-based LoRaWAN to ensure the whole network is not vulnerable to several attacks and intrusions [1–12]. Due to the resource-constrained LoRaWAN devices, heavy and complex cryptographic models are not practical solutions to improve the security level of the existing LoRaWAN [1–12]. Therefore, the LoRaWAN security model is fully implemented using the AES 128 bit symmetric encryption with different modes of security such as the Cipher-based Message Authentication Codes (CMAC) mode for data integrity, and the Counter (CTR) mode for data encryption and decryption [6]. Moreover, each security key is derived for its rightful purpose such as the Application Key (AppKey), which is pre-shared and only known between the end device and the Network Server (NwkS) when the device is manufactured. The AppKey is used to generate two session keys: AES 128 bit Application Session Key (AppSKey) and Network Session Key (NwkSKey). The AppSKey is shared between the end device and the Application Server for the encryption and decryption of the payload, while NwkSKey is shared between the end device and the network server for initiating the communication. NwkSKey is also used for message integrity using Message Integrity Check (MIC) [6]. Moreover, LoRaWAN devices are activated for communication using either Activation By Personalization (ABP) or Over The Air Activation (OTAA) [7]. In OTAA, the activation is initiated from the end devices side by sending a join request (JR) message to the NwkS, where the message is composed of the 64 bit hexadecimal Device EUI (DevEUI) device universal identifier, and the 64 bit hexadecimal Application EUI (AppEUI), AppS universal identifier and a 2 octets random number as Device Nonce (DevNonce) used for integrity [7]. The AES 128 bit AppKey is pre-programmed and pre-shared between the NwkS and the end devices, in place of the encryption of the JR message; the AppKey is responsible for computing the MIC of the transmitted message [6]. If the JR is approved, the NwkS responds with the join accept (JA) message composed of the 32 bit hexadecimal Device Address (DevAddr), Network Identifier (NetID), and the 3 octets Application Nonce (AppNonce). In ABP, the NwkS and the end-device share transmit the JR and JA the message as in OTAA, the only difference is the pre-programmed and pre-shared of the AppSKey, the NwkSKey, and the device address (DevAddr) between the servers and the end devices before communication to ensure that the entities are ready communication and equipped with relevant keys [6,7,9,11,12].

Currently, several LoRaWAN's security key models have been proposed and implemented [2,4,5,7] in the literature to strengthen the defense against attacks and intrusions. However, these proposed models are not effective against unknown attacks and intrusions. Therefore, this paper designed and implemented two Trusted Key Management Server (TKMS) Algorithms A and B, which deployed a trusted key server to securely generates and manages the encryption keys in the network while being distributed across the end device of the network server. Algorithm B is the improved Algorithm A based on its security shortfalls and produces a novel model based on several contributions to knowledge in this paper as follows:

i. We provided a comprehensive analysis of the existing LoRaWAN's security key models to analyze their strengths and weaknesses.
ii. We designed and implemented two Trusted Key Management Server (TKMS) Algorithms A and B, where B is an enhanced A.
iii. We applied the formal analysis methods to prove logic correctness and to verify the proposed models against all possible LoRaWAN attacks by improving the security level of the trusted key server. The model proposed in [7] though achieved confidentiality, integrity, authentication, and security against replay and DoS attacks; however, based on the results of the security claims verification in terms of secrecy, alive, weakagree, nisynch, niagree, it was discovered that the entities were only secured within the bounds. The network should be secured within bounds for any possible internal attacks, and outside bounds for possible external attacks, and if the model is

only secured within bounds, external attacks are likely to exploit the network as no security is put in place for eliminating them.

We enhanced the security of the model proposed in [7] by applying the following:

i. Trusted key server and AppKey independency: A trusted key server is implemented to reduce the possibilities of DoS and server attacks. As discussed in the analysis of existing security models, the AppKey is pre-distributed between the end-device(s) and the network server or the trusted third party; however, this could lead to a replay attack, denial of service (DoS) attack, or server attack if the AppKey is exploited before initiating the communication. In this paper, the implemented trusted the third party generates the AppSKey and the NwkSKey after receiving the AppKey from the entity that transmitted the request to join messages; this ensures a unique AppKey is activated for every request to join a session. Moreover, for every request to join sessions, AppKey generates the NwkSKey and AppSKey using uniquely generated prime numbers.
ii. Replay attack countermeasure: A timestamp is generated for every transmitting entity in the request to join sessions and stored in the receiving entities for validation during accepting to join sessions. This ensures that every transmitted message is freshly generated. If the received timestamp is not the same as the previous timestamp of the very same transmitting entity, then there is a possibility of a replay attack.
iii. Authentication and message integrity: The existing models employ symmetric encryption, asymmetric encryption or even both for authentication and validation of message integrity; however, these algorithms are heavy on operations. In this paper, we implemented hashing algorithm for performing authentication and validating message integrity as they have low processing and computation times.
iv. Logic correctness: The design of LoRaWAN security models is different based on different researchers' objectives. However, the LoRaWAN security models are common in validating and proving the authenticity of the entities. With most of the existing works in LoRaWAN security using security verifying tools only for security claims analysis, authenticity is discussed without proof based on the security claims. In this paper, the Scyther tool is used for security claim analysis, and BAN logic formal analysis is used to prove the logical correctness of our proposed model including the validation of entity authenticity. This is to guarantee the receiving entity that the received message is from the trusted and authenticated entity.
v. Prime numbers: Prime numbers as they are easy to compute but difficult to break, maximize the security level of the entities in two operations and they are uniquely generated for each operation. First, they are used for entity authentication instead of nonces as these nonces are unsecured pseudo-random numbers, second, they are used for generating NwkSKey and AppSKey; this counter various key attacks as the prime number are resource constraining to be guessed before exploiting all the parameters used to generate the key.

The rest of this paper is organized as follows: Section 2 presents the background on symmetric encryption and hashing, Section 3 presents the related works and their security shortfalls, Section 4 is the methodology, Section 5 is the detailed discussion of the proposed solutions, and Section 6 presents the security analysis, while Section 7 presents this paper's discussions. Section 8 is this paper's conclusion.

2. Symmetric Encryption and Hashing

This section presents the background on security algorithms applied in this paper: symmetric encryption and the hashing technique, which are categorized under cryptography, where the strength of any transaction is measured based on its key. A longer key guarantees strong security and is important the keys are always secured as their secrecy guarantees the security of the system.

2.1. Symmetric Encryption

Symmetric encryption algorithms are referred to as the secret key encryption algorithms due to using a single key kept securely within the system, where this key is also used during the encryption process and the decryption process [13,14]. As compared to its counterpart asymmetric encryption, which uses two keys for encryption and decryption processes, symmetric encryption in most cases is considered to be more secured and very fast in computations and transmissions [13–15]. Symmetric encryption is often more suitable than asymmetric encryption when securing a large amount of data. Moreover, for a transaction to be initiated when using the symmetric scheme, the keys are distributed across the participating systems. It can be a daunting task to manually employ a single key that is shared over a network; however, copying this key from a centralized key entity is a possible solution, where an administrator needs to ensure that the scripting and policy of copying these keys are fully functional always [13,14]. Symmetric encryption can be a stream cipher or a block cipher. A stream cipher uses a stream of a symmetric key for encryption and has a faster computation than block ciphers as they use a simple mathematical computation [13]. A block cipher encrypts blocks of data one at a time [14,16,17]. The encrypted block is usually of the same length as unencrypted data [14,16,17].

Different symmetric encryption algorithms are being employed in cryptography such as the following to name a few:

i. Advanced Encryption Standard (AES): The AES algorithm can be implemented with three different schemes, AES 128, AES 192, and AES 256, where 128, 192, and 256 are the key lengths in bits. AES gained popularity for implementation as it is very secured and very fast [14–18].
ii. Data Encryption Standard (DES): The DES algorithm's original standard uses a 56 bit key with the combination of 0s and 1s. Some years back, it was difficult to break DES due to the computers that were used; however, in modern society, powerful computers have been developed which can take a lesser time for breaking the DES algorithm. Based on this, DES has been discontinued in most implementations [14–18].
iii. Triple Data Encryption Standard (3DES): The 3DES is an advanced and improved version of the DES algorithm, where it applies the DES operations and functions three times. The strength of 3DES lies in a longer key than DES by generating a 56 bit key three times. The key is to be fully specified for all three encryption iterations. However, there is an option of using the key same for all three iterations which will be a 56 bit key or the same key for two iterations and the iteration with a different key which will be a 112 bit key, or all three iterations use different keys summing up to a 168 bit key is used [14–18].
iv. International Data Encryption Algorithm (IDEA): The IDEA is a 128 bit key algorithm that was developed aiming to replace the DES algorithm. The shortfall in implementing IDE was based on two reasons, it is prone to produce a range of weak symmetric keys, and many symmetric algorithms are faster than IDE but have the same security level as IDEA [14–18].

2.2. Hashing Function

In cryptography, hashing functions or algorithms are based on scrambling the data using the hash table index [19–21]. The hash function takes accepts arbitrary data as the input and applies a mathematical function to compute a hashed value of a fixed length [15–17]. Moreover, hashing has several key principles; first, hashing is a one-way function, where the hashed value cannot be reversed to generate arbitrary data, second, hashing should be collision free, where a hashed value should not be produced from different sets of arbitrary data, and third, the generated data cannot be changed without having to change the hashed value [18–20]. Popularly deployed hashing algorithms are:

i. Secure Hash Algorithm-version 1 (SHA-1): The SHA-1 hashing algorithm produces a 160 bit of hashed value. The SHA-1 gained popularity as an alternative to MD5 by eliminating most of the weaknesses that surfaced from MD5 [19–21].
ii. Secure Hash Algorithm-version 2 (SHA-2): The SHA-2 is a suite of several hashing algorithms of SHA-224, SHA-256, SHA-256, SHA-384 and SHA-512, where the numbers represent the output length in bits [19–21].
iii. Message Digest-version 5 (MD5): MD5 is a hashing algorithm that produces a digested value of 128 bits; however, it has shortfalls in terms of more collisions surfacing [19–21].

3. Related Works

This section presents the analysis and limitations of some of the existing works on security models in LoRaWAN to improve the security level of these models. Due to the resource-constrained nature of the IoT platform such as the limited battery capacity and memory, implementing a heavy security model is not feasible for IoT-based applications. The authors in [1] argue that an existing Lightweight Encryption Algorithm (LEA) standardized for IoT devices was prone to side-channel attacks that exploit the consumed energy. However, the authors proposed an improved LEA algorithm by employing an arbitrary value instead of the masking technique to eliminate differential side-channel attacks which minimizes the processing time. In a similar study, to address the high energy consumption of AES in LoRaWAN, Tsai et al. [11] proposed a Low Power and Highly Secure Communication Scheme named (SeLPC) that minimizes the overall energy consumption by at least 26% due to the reduction in high data encryption power and the AES encryption cycles. Moreover, the proposed model is secured against known key attacks, eavesdropping attacks, and replay attacks. Routsalainen et al. [9] investigated a wireless key generation for LPWAN-based applications and discovered feasibility in long-distance communications, deep in buildings, and session keys that have high randomness. The study was geared towards addressing challenges such as limited channel occupation, lengthy payloads, duty-cycled transmissions and receptions.

In the same vein, Kim et al. [4] also suggested a Dual Key-based generation and update model in the essence of improving the key privacy and security of the existing model due to cryptoanalysis key attacks. The model is based on AppKey and NwkKey which are independently and separately generated to produce AppSKey shared between the end device and the application server, and the NwkSKey shared between the end device and the network server, respectively. The proposed model also improves key security and privacy based on its independent and automated session keys generation for each communication layer. Similarly, Roselin et al. [8] proposed a Lightweight Authentication Protocol (LAUP), which employs symmetric encryption without using pre-shared keys. It uses several techniques: a session keys sharing establishment using information history, and a four-flight authentication establishment approach [8]. It was designed to enhance LPWAN's security against other asymmetric encryption-based models considered resource constrained with the session keys' pre-sharing vulnerable to attacks.

Several researchers argued that adding a trusted third party to the LoRaWAN security model could strengthen its security in terms of key generation and management and minimize complex operations. Accordingly, Naoui et al. [7] proposed an improved LoRaWAN security model based on a trusted third party to manage keys in the network between the entities. Several countermeasures to address all possible attacks against LoRaWAN were implemented such as a trusted third party to generate and manage session keys in the network without being exposed to the public, avoiding DoS attacks, and use of a timestamp to avoid exhausting the memory of LoRaWAN entities. Moreover, the trusted third party computes the NwkSKey and AppSKey random numbers instead of the network server generating the session keys and renewing the AppKey every session to avoid replay attacks. In a similar work, Tsai et al. [10] argued that a trusted third-party key exchange protocol could raise several security concerns related to the inflexible,

inefficient and unreliable protocol. The authors proposed a highly efficient multi-key exchange protocol based on current time encryption, a 2D operation, and elliptic curve cryptography [10]. It was designed for efficient session keys exchange, improved data encryption of sensitive data, mutual authentication, and a shorter processing time of approximately 3.78-fold faster than the existing models, and improved number of session keys generation by approximately 40 keys in a single process [10]. The model is also secured against forgery, impersonation, eavesdropping, and replay attacks. Han et al. [2] also suggested an enhanced security model using root key management, a lightweight security model deploying a Lightweight Rabbit Stream Cipher Based-Key Derivation Function (KDF) approach to reduce key processing times in terms of reduced key ransom generation, deviation time, and frequent root key updates. The model enhances the basic security of LoRaWAN 1.1 specifications but is prone to a cryptoanalysis of security key attacks. proposed model is a.

However, the security shortfalls of the model discussed above are as follows: In [1], limited memory as one of the key issues in IoT was not considered as generating additional 4 byte data can affect the memory usage of the devices since the data are stored for encryption and decryption. In our proposed algorithms, we consider memory usage by implementing hashing algorithm for message integrity and authentication as it has a shorter computational time and lower memory usage than symmetric encryption. Additionally, the authors of [11] only studied application data security and the NwSKey used by SeLPC in the MAC layer to generate the MIC with no periodic updates. However, in our proposed algorithm, there is key independence for message integrity calculations and authentication using AppSKey and NwkSKey for end device and application server communication and end device and the network server communication, respectively. Again in [9], refreshment periods occur after a while, which could affect the whole network if old keys are being exploited before being refreshed. In our proposed models, the keys are refreshed for every new request to join a session and when the entity time stamp expires.

In the model proposed by Kim et al. [4], the pre-programming and pre-loading of NwKey and AppKey lacks impracticable security and the whole network can be breached if the end nodes are exploited. Using nonces in end-nodes, applications and network servers affect the memory usage and occupation as these nonces are stored. Flooding attacks lead to DoS on the end nodes to extract the session key due to the pre-programming and pre-loading of these session keys. A new session key should be generated for every join request to avoid using old session keys to generate new session keys as attackers can use brute force attacks to exploit these old session keys. However, in our proposed algorithms, only the AppKey is pre-programmed into the end device and shared with the key server as a parameter during request to join procedures. Similarly, Roselin et al. [8] proposed a model that is limited for validation against three attacks only, though it might be based on the objectives of the proposed work, other existing security attacks should be analyzed and validated. To overcome this limitation, in our work, we evaluated the proposed algorithms against all possible LoRaWAN attacks; this was to be familiar and aware of the attacks that might affect our proposed model in future. Additionally, Naoui et al. [7] model used unsecured random numbers that are generated to compute two different session keys and the NwKey is not changed in every session as the AppKey. However, this paper proposed the use of prime numbers instead of nonces as they are randomly generated and not secured, whereas prime numbers are easy to compute but difficult to reserve. This brings about a daunting challenge for the intruders if they tend to intrude the network, they will need to know the combination of the prime numbers used to generate the actual prime numbers. Moreover, for authentication and re-calculations of the message integrity after transmission, we implemented hashing instead of symmetric or asymmetric encryption; hashing has low computational cost and time. Again, Tsai et al. [10] lack minimizing the power consumption of mobile devices, which is key to developing THMEP for heterogeneous systems as some of the operations performed are too complex for resource-constrained sensor nodes. However, in our proposed model, a hash is

implemented for authentication and calculations of message integrity as it has a lower computational time and lower memory usage than symmetric encryption. Lastly, in Han et al. [2] model, pre-sharing and distribution of the root keys between the end devices and the join server could lead to memory overheads in the end devices as they have limited memory. This could in turn, lead to flooding attacks that could exploit the end devices for stealing the root keys. This is addressed in our model by pre-programming only the AppKey into the end device, where it is transmitted to the key server during the request to join procedures to use it for generating session keys.

4. Methodology

To analyze and verify the security efficiency of a security algorithm, we applied formal analysis methods using security protocol verifying tools, logic proofs and rules for model correctness. The tools are based on formal analysis methods under the assumptions of perfect cryptography, where all the implemented cryptographic functions are assumed to be perfect, and the logic proofs are based on proving model correctness. This is based on the concept that the intruders do not know the transmitted data unless the encryption key is exposed. Researchers adopt the use of these tools to find all possible security attacks within the proposed models while being developed. Though many sceptics may arise from this tool analysis, in many cryptographic practices, Scyther can prove many security protocols to be correct or possible attacks can be found. After successful modelling of the security algorithms, this security tool accepts a high-level input description or language of the proposed security algorithms and performs a security verification using several security properties or claims. This tool is designed to find the best or all possible attack paths that can be used by intruders to exploit the network and the participating entities. Moreover, BAN logic is applied for reasoning with the proposed algorithms cryptographically based on logic correctness.

Moreover, to model security algorithms in Scyther, the algorithm is designed using communicating entity roles, where each entity adheres to certain transmission and reception events for communication. Furthermore, several security key algorithms such as symmetric and asymmetric encryption, and hashing can be employed to securely distribute security keys and encrypt the data between the communicating entities [22–25]. The Scyther verifying tool is based on a series of security claims for verifying the security level of the proposed algorithm.

(1) Secrecy claim: All the transmission parameters between the participating entities in the network are to be secured at all times [22–25].
(2) SKR claim: The SKR claim resembles that of the Secrecy claim if not being revoked; however, the SKR claim is used to mark employed session keys[22–25].
(3) Empty claim: The empty claim is used by Scyther to identify that a certain will be ignored for verification. The applicability of this claim extends to employing Scyther as a back-end verification tool for other verification means [22–25].
(4) Reachable claim: This claim is used to verify that the specific claim is reachable; it is indicated by at least one trace pattern if it exists. This claim is also used to verify if there are existing errors within the specification of the modelled protocol[22–25].
(5) Alive and weakagree: For this claim, all the entities are implemented with the same scheme; alive claims ensure that all the transmitted messages between the entities are encrypted with the same scheme, and weakagree ensure that all the participating entities are running on the scheme [22–25].
(6) Non-injective agreement (niagree) claim: All the participating entities in the same network should adhere to and agree on the same parameters to use for their transmission [22–25].
(7) Commit, running claim: This claim is another form of niagree, where a niagree for defining a specific role within a set of data by adding relevant signal claims. The Commit claim is invoked to identify the effective claim of the protocol, and the

Running claim ensures that the effective claim invoked by Commit attains correctness of the existing Running signal within the found trace [22–25].

(8) Non-injective synchronization (nisynch) claim: All the transmission processes and sessions to take in the network between the entities are to adhere to all the security specifications of the proposed protocol and all the participating entities shall adhere to being synchronized in their current state [22–25].

Based on these properties, we employed the formal analysis and validation method to design, develop and evaluate the proposed improved TKMS Algorithm B in LoRaWAN. A high-level flowchart in Figure 2 is used to illustrate the depth of enhanced TKMS Algorithm A and its security analysis against all possible LoRaWAN attacks.

5. Proposed LoRaWAN Security Model

5.1. Overview

In practice, LoRaWAN's security model proffers only a minimum-security level for the network as it deploys encryption keys to secure the communicating entities, shared data, and even the protection of the encryption keys as well against intruders. However, the network could be vulnerable to key and data attacks that can adversely affect it. Although several proposed models for improving the LoRaWAN key management security models exist such as in [2,4,6,7,10,11], it is important to consider the characteristics and the attack' nature when designing and developing an improved LoRaWAN key security model. With several parameters involved when generating and securing both the communicating keys and the parameters, all are considered data and should be secured with encryption keys that are securely managed. With an intruder having sufficient knowledge about possible weaknesses and shortfalls of the LoRaWAN security model, several attacks can be launched to compromise the network. For instance, attacks such as replay attacks [7,10] and the man in the middle attack [25]. Moreover, the intruder may perform the bit-flipping attacks [26] by altering the bits of the transmitted causing the receiving entities not to decrypt the encrypted transmitted data. This paper addresses such security challenges in the LoRaWAN by designing and implementing two security models using the Scyther security verifying tool. The aim is to enhance the key server to efficiently generate and manage keys securely. However, the proposed models do not accommodate the communication between the end devices, but rather demonstrate the efficiency of key generation and distribution in the LoRaWAN network by securing the entities and the network before any communication may occur between the end devices. Moreover, the application server is excluded since it is only active during communication. However, if the end devices are to communicate, the AppKey shall be shared between the end devices and the application server.

The key distribution of the proposed model between the entities is presented in Figure 1. The two TKMS algorithms are Algorithms A and B, where Algorithm B is an enhanced Algorithm A. The end device in both models is activated via OTAA rather than ABP due to security issues with pre-programmed keys. As shown in Figure 1, the three main entities and three keys being generated and distributed across the entities by the trusted key server in the proposed key distribution architecture are as follows:

i. End-device: A LoRaWAN device that is being deployed in the network after manufacturing. This end device is pre-programmed with several parameters such as the AppKey, the device universal identifier used to identify itself in the network, and the AppKey that will be shared with the trusted key server during the request to join process.

ii. Trusted key server: The trusted key server is employed as a trusted party to generate and manage the keys securely and efficiently in the network. It is used in our model to ease some of the complex operations that were performed by the network server alone such as key generation and update. This server upon receiving AppKey for the end device, generates two session keys, the AppSKey used for securing the

communication between the end device and the application server, and the NwkSKey used for securing between the end device and the network server.

iii. Network server: This is responsible for generating and distributing network parameters with the end device so that during transmission the end device is knowledgeable of the network to transmit on, and the unoccupied channels that can be used for the transmission.

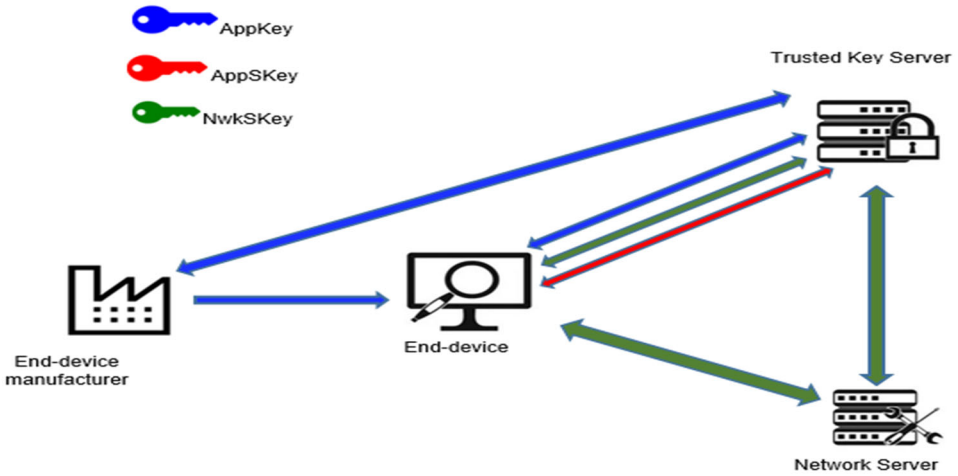

Figure 1. Key distribution architecture.

5.2. TKMS Algorithm Design

Using the Scyther security verifying tool, the TKMS Algorithm A has been designed, developed, and evaluated against all possible security attacks that are likely to affect the LoRaWAN. Moreover, the Scyther tool has been used to perform security evaluations and validations against all possible existing attacks in LoRaWAN. As depicted in Figure 2 and discussed in Section 4 of the proposed solution, it is shown that two models have been designed and developed, the TKMS Algorithms A and B. Scyther is used for security validation as follows:

(a) In verifying the protocol there were no end-device attacks detected on the TKMS Algorithm A, and as for the network server and the key server, there were no attacks within the bounds. However, to fully secure the network, security should be enforced against external attacks outside bounds and for internal attacks within bounds.

(b) In verifying all security automatic claims for each entity, we realized that though the model has a certain level of security, the end device remains secure against all possible attacks in LoRaWAN, but the network server and the key server are still secure against attacks within bounds only.

After a series of evaluations and validations performed approximately seven times with the Scyther tool, TKMS Algorithm A was modified to Algorithm B conforms to the desired security of the Scyther tool and is secured against all possible attacks in LoRaWAN both within bounds and outside bounds.

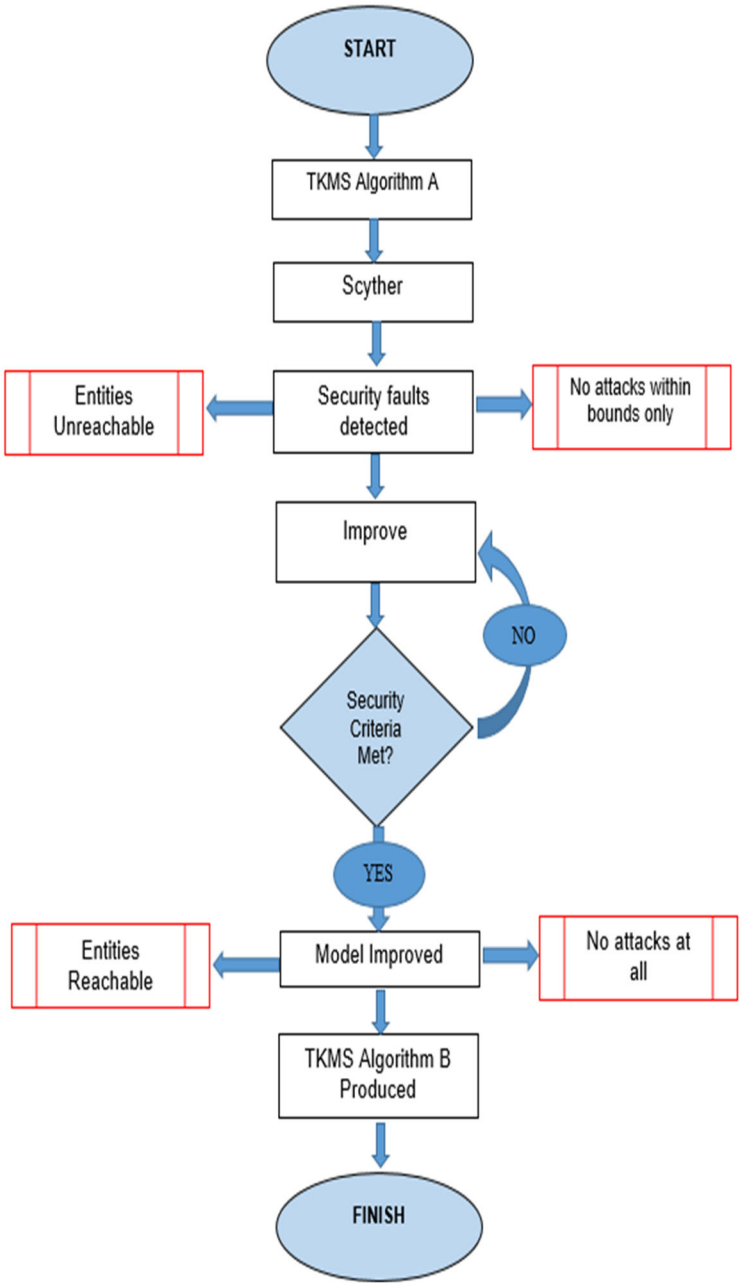

Figure 2. Key generation and distribution process in TKMS.

The mathematical formulas of notations in Table 1 are as follows:

- NwkSKey = $AES_{128}(AppKey, 0x1|KeySvrPrN1|KeySvrT|JoinEUI|pad_{16})$, $JoinEUI$ are composed of the transmission parameters for a specific transmission;
- AppSKey = $AES_{128}(AppKey, 0x2|KeySvrPrN2|KeySvrT|JoinEUI|pad_{16})$, $JoinEUI$ are composed of the transmission parameters for a specific transmission;
- Prime numbers ≈ $a_1 * a_2$, ∀ a ∈ $a_{1,\ldots,}$ $\sqrt{a}$, ∴ a is prime IFF ∃ a: $a < n$, then $a \neq$ prime if $a \geq n$;
- CFList = $\{CFList_{[0]}, \ldots, \}$ represent free channels;
- H (•) = $H(h) \approx a = H(b)$, cannot predict b from a hashed value a such that $a = H(X_{bits} \oplus Y_{bits})$, choosing a value or data of X_{bits} on a scale of 1 to 10^{24} bits to produce a hashed value a, it is difficult to predict the data of Y_{bits};
- NwkID = $\{NwkID_{[0]}, \ldots, NwkID_{[i]}\}$ represent the identity of the network;
- Timestamp (T_i) = $T_i \in \{T_x - T_y > \theta\ T\}$ is accepted as a valid timestamp otherwise a replay attack is detected;

- Message (M) encryption = $AES_{128}(E_K(E_x, E_y)|PrimeN_e|T_x|T_p|pad_{16})$, where E_x, E_y are the entities EUI's, $PrimeN_e$ are the prime numbers of the entities, T_x is the timestamp of the entity and T_p are the transmission parameters.

Table 1. Notation and definitions.

Notations	Definitions
KeySvr	Key Server (or TKMS) Universal Identity
EDs	End Device Universal Identity
NwkS	Network Server Universal Identity
k(EDs, KeySvr)	Encryption session key between EDs and KeySvr
k(KeySvr, NwkS)	Encryption session key between KeySvr and NwkS
k(NwkS, EDs)	Encryption session key between NwkS and EDs
H	Hash function
ε	Encryption between the entities (EDs, KeySvr, and NwkS)
AppKey	Application Key
AppSKey	Application Session Key
NwkSKey	Network Session Key
EDsT	End Device Timestamp
KeySvrT	Key Server Timestamp
NwkST	Network Server Timestamp
EDsPrN	End Device Prime Number
NwkPrN	Network Server Prime Number
KeySvrPrN	Key Server Prime Number between NwkS and KeySvr
KeySvrPrN1	Key Server Prime Number between EDs and KeySvr for generating NwkSKey
KeySvrPrN2	Key Server Prime Number between EDs and KeySvr for generating AppSKey
CFList	Channel List
RxDelay	Transmission Delay
NwkID	Network Identity
DevAddr	Device Address

Moreover, in both algorithms, the NwkSKey reinforces the message integrity of the transmission parameters between the EDs and the NwkS, whereas the AppSKey reinforces the message integrity of the transmission parameters between the EDs and the AppS. In this paper, the communication between the EDs is not within the scope, where for the EDs to communicate, the AppSKey should be firstly distributed and validated between the AppS and the transmitting end device for message integrity. Moreover, the combination of prime numbers (PrN) is used for authentication and message integrity instead of the nonces as they are randomly generated without any being secured. Based on cryptography, the computation of prime numbers is easy but difficult to reverse. In both algorithms from Step 1 to Step 3, encryption session keys are used to initiate the join request sessions between the entities, and from Step 4 to Step 11 hashing is used for entity authentication, message integrity, and faster computation as they have fewer computation times than symmetric and asymmetric encryption during authentication and message integrity calculations. Furthermore, each entity generates its timestamp for every new request to join sessions to guarantee freshly generated messages without a replay attack possible. The entities may alternate the stored parameters for authentication and message integrity calculations, this is to avoid intruders guessing the parameters for the next transmission.

5.2.1. TKMS Algorithm A

This subsection presents the Algorithms 1 and 2: TKMS Algorithm A to demonstrate the efficiency of key generation and distribution amongst the entities. The TKMS use the received AppKey to generate two session keys; the AppSKey is used for securing the communication between the application server and the end device should the end devices

wish to communicate, and the NwkSKey is used to secure the communication between the network server and the end devices. Symmetric encryption is used to initiate communication between the entities and ensure the transmission parameters are well secured during the request to join procedure initiated by the end device, whereas hashing is used for authentication and calculations of message integrity.

Initialization Phase: Initially, as presented in Figure 2, the EDs are pre-programmed with the AppKey, KeySvr identity and NwkS identity from the manufactures side when they are being designed. All other keys (AppSKey, and NwkSKey) are generated and distributed by the trusted KeySvr upon receiving the JR message from the end device, where both the AppSKey and the NwkSKey are shared with the end device for message integrity, and the NwkSKey is also distributed to the AppS.

Algorithm 1: TKMS Algorithm A

Input = $(\{M\}_\varepsilon|\{m\}_H)$ *//Request To Join Message: Encrypted and Hashed message*
Output = $(\{m\}_H)$ *//Accept To Join Message: Hashed message* Steps:
WHILE (Request to Join Initiated) {
IF $[(\{M\}_\varepsilon|\{m\}_H)]$ $is\ transmitted\ between\ entities$) {
THEN
1st: EDs ⇒ KeySvr { $[(\{M\}_\varepsilon|\{m\}_H)]$}
2nd: KeySvr ⇒ NwkS { $[(\{M\}_\varepsilon|\{m\}_H)]$ }
3rd: NwkS ⇒ EDs {$[(\{M\}_\varepsilon|\{m\}_H)]$ } }
IF ($(\{m\}_H)$ $is\ transmitted\ between\ entities$) {
4th and 5th: EDs ⇔ KeySvr { $(\{m\}_H)$ }
6th and 7th: EDs ⇔ NwkS { $(\{m\}_H)$ }
8th: EDs ⇒ KeySvr { $(\{m\}_H)$ }
9th and 10th: KeySvr ⇔ NwkS { $(\{m\}_H)$ }
11th: KeySvr ⇒ EDs
{ $(\{m\}_H)$ } }
IF $[(\{M\}_\varepsilon|\{m\}_H)\ OR\ (\{m\}_H)]$ *is approved by receiving entity until the 11th step*) {
//Authentication & MIC approved
Request To Join Approved }
ELSE {
"Terminate Request To Join = "Unauthorized user found and Message Falsified" }
}

Join and Key Distribution Phase: The steps involved are discussed as follows:

- Step 1: The EDs initiate the system by sending the request to join message (EDsPrN || EDs || NwkS || EDsT || AppKey) to the KeySvr. The message is composed of the uniquely and freshly generated parameters of; EDsPrN, EDs, EDsT and the AppKey; these parameters are encrypted with the encryption session key (k(EDs, KeySvr) of the EDs and the KeySvr. Moreover, the same unique parameters are signed with a H function as [(H(EDsPrN || EDs || NwkS || EDsT || AppKey)].
- Step 2: Upon receiving the request to join the message from the EDs, the KeySvr performs decryption on the message using k(EDs, KeySvr), authentication and re-calculating message integrity as [(H(EDsPrN || EDs || NwkS || EDsT || AppKey)]. If valid, the KeySvr stores the received parameters of EDsPrN, EDs, EDsT and the AppKey, then forwards the request to join message to the NwkS (EDsPrN || KeySvrPrN || EDs || KeySvr || KeySvrT) encrypted with k(KeySvr, NwkS) by generating a unique KeySvrPrN, and KeySvrT, and also hash the message as H (EDs || NwkPrN || DevAddr || RxDelay || CFList || NwkID || NwkST)].

- Step 3: If the NwkS successfully receives the message, the NwkS performs decryption on the received encrypted message using k(KeySvr, NwkS), authentication and re-calculates the message integrity as [H(KeySvrPrN || EDs || KeySvr || KeySvrT)]. If valid, the NwkS stores all the received parameters in the message, then directly send the accept to join message to the EDs as [(EDsPrN || NwkPrN || DevAddr || RxDelay || CFList || NwkID || NwkS || NwkST || KeySvr || KeySvrT)] encrypted with (NwkS, EDs) with uniquely generated DevAddr, CFList, RxDelay, NwkID, and NwkPrN, and also hash the message as [H(EDs || NwkPrN || DevAddr || RxDelay || CFList || NwkID || NwkST)]; Accept to join message is directly transmitted to the EDs using a unique NwkID that will ensure that requesting and accepting channels are isolated in the network. This is to avoid any channel attacks. DevAddr is assigned to the specific EDs that initiated the request.
- Step 4: If the EDs successfully receives the message, the EDs performs decryption on the received encrypted message using k(NwkS, EDs), authentication and re-calculates message integrity as [H(EDs || NwkPrN || DevAddr || RxDelay || CFList || NwkID || NwkST)]. If valid, the EDs store the uniquely generated parameters in the message, and send a hashed message [H(DevAddr || NwkS || NwkID || KeySvrT || EDsPrN)] to the KeySvr. DevAddr uniquely identifies the specific end device to receive the AppSKey, NwkSKey, and other important parameters such as unique NwkID for transmission, and the prime numbers used to generate and hash the AppSKey and NwkSKey.
- Step 5: The KeySvr will only perform authentication and re-calculates message integrity as [H(DevAddr || NwkS || NwkID || KeySvrT || EDsPrN)] if the hashed message is successfully received. If successfully received from the EDs, the KeySvr stores the DevAddr and NwkID, then send a hash message to the EDs as [H(DevAddr || KeySvrT || NwkSKey || KeySvrPrN1 || NwkID) || H (DevAddr || KeySvrT || AppSKey || KeySvrPrN2 || NwkID)]. The AppSKey and NwkSKey are uniquely generated from AppKey with unique KeySvrPrN1 and KeySvrPrN2 and will be used by the EDs should they wish to communicate with other EDs.
- Step 6: Upon a successful reception, the EDs will then authenticate and re-calculate the message integrity with [H(DevAddr || KeySvrT || NwkSKey || KeySvrPrN1 || NwkID) || H (DevAddr || KeySvrT || AppSKey || KeySvrPrN2 || NwkID)] and stores KeySvrPrN1, KeySvrPrN2, NwkSKey, and AppSKey, then send the hashed request to acknowledge message to the NwkS as [H(KeySvr || EDsPrN || NwkPrN)] to notify the NwkS that the session keys and prime numbers are successfully received from the KeySvr and request the NwkID to be verified before starting any communication with other end devices. The EDsPrN and NwkPrN are used by EDs for authentication, and the KeySvrPrN1 and KeySvrPrN2 are used by the EDs to generate its AppSKey and NwkSKey from the AppKey and check if they are the same as the ones received from the KeySvr, if different, the ED terminates the request.
- Step 7: If the NwkS successfully receives the hashed acknowledgement request from the EDs, the message integrity is re-calculated as [H(KeySvr || EDsPrN || NwkPrN)] and if successful, an acceptance to acknowledge hashed message is generated and transmitted as [H(DevAddr || NwkST || NwkID)], with the NwkId to identify the network to use for future communication within the same request to join.
- Step 8: If the hashed message is received by the ED, the ED will authenticate and re-calculate the message integrity as [H(DevAddr || NwkST || NwkID)], if successful, then the ED will generate and forward an accept to acknowledge hashed message as [H(NwkS || NwkID || KeySvrT)] to the KeySvr to notify the KeySvr that request has been acknowledged, the NwkID is verified and the NwkSKey can be transmitted to the NwkS.
- Step 9: The KeySvr receives the hashed message, authenticates and re-calculates the message integrity as [H(NwkS || NwkID || KeySvrT)] and if successful, then

generates a unique NwkSKey, and transmits a hashed message to the NwkS with the NwkSKey as [H(KeySvrT || NwkSKey)].

- Step 10: The NwkS receives the hashed message in Step 9, authenticates and re-calculates the message integrity from the received hashed message as [H(KeySvrT || NwkSKey)], if successful, NwkSKey is stored and a hash message of [H(KeySvrT || KeySvrPrN)] is generated for notifying the KeySvr that the NwkSKey is successfully received.
- Step 11: The KeySvr receives the hashed message in Step 10, authenticates and re-calculates the message integrity from the received hashed message as [H(KeySvrT || NwkSKey)], if successful, the KeySvr generate a hashed message of [H(KeySvrT || NwkID || EDsT)] for notifying the EDs that the join procedure is successful.

Algorithm 2: Enhanced-TKMS Algorithm

Input = $(\{M\}_\varepsilon|\{m\}_H)$ *//Request To Join Message: Encrypted and Hashed message*
Output = $(\{m\}_H)$ *//Accept To Join Message: Hashed message*
Steps:
WHILE (Request to Join Initiated) {
IF $[(\{M\}_\varepsilon|\{m\}_H)]$ $is\ transmitted\ between\ entities$) {
THEN
1st: EDs ⇒ KeySvr { $[(\{M\}_\varepsilon|\{m\}_H)]$}
2nd: KeySvr ⇒ NwkS { $[(\{M\}_\varepsilon|\{m\}_H)]$ }
3rd: NwkS ⇒ EDs
{$[(\{M\}_\varepsilon|\{m\}_H)]$ } }
IF ($(\{m\}_H)$ $is\ transmitted\ between\ entities$){
4th and 5th: EDs ⇔ KeySvr { $(\{m\}_H)$ }
6th and 7th: EDs ⇔ NwkS { $(\{m\}_H)$ } //Enhanced Steps for authentication and message integrity
8th: EDs ⇒ KeySvr { $(\{m\}_H)$ }
9th and 10th: KeySvr ⇔ NwkS { $(\{m\}_H)$ }
11th: KeySvr ⇒ EDs
{ $(\{m\}_H)$ } }
IF $[(\{M\}_\varepsilon|\{m\}_H)$ OR $(\{m\}_H)]$ *is approved by receiving entity until the* 11^{th} *step*) {
//Authentication & MIC approved
Request To Join Approved }
ELSE {
"Terminate Request To Join = "Unauthorized user found and Message Falsified" }
} EDs receive a message in *Step 11* and performs authentication and message integrity check, if approved, then Join Procedure complete }
//Request to Join Procedure Complete

5.2.2. TKMS Algorithm B

To improve the security of Algorithm A, the following steps have been modified to produce an enhanced TKMS algorithm, Algorithm B. The enhancement starts from Steps 3 to 9 of Algorithm A as follows:

Step 3: The hash message transmitted to the ED by the NwkS is being modified. The EDs and NwkPrN parameters have been replaced with the EDsPrN; this is to avoid attacks such as bit flipping, where if the whole message is intercepted, then parameters of the encrypted message can be used to reverse the hash message as the same parameters are used in both messages. Moreover, the EDsPrN can be used by the ED upon receiving the message for ensuring that its authenticity has not been altered.

Step 6: The NwkPrN is replaced with the EDsT for ensuring the NwkS that the request to acknowledge message is freshly generated and a NwkID transmitted in Step 7 is safe to be transmitted to allow any future communication likely to occur in the request to join can take place for the ED which generated the EDsPrN.

Step 7: Replace NwkST with NwkPrN to add more security to NwkS authentication using prime numbers.

Steps 9 and 10: A NwkID is being added to the hash function to ensure that the KeySvr and NwkS shared the same NwkSKey on the same network and the message is freshly generated using KeySvrT.

5.2.3. BAN Logic

In this section, we discuss proving the security efficiency and reliability of our proposed Algorithm B, which is an improvement of Algorithm A by applying BAN logic for logic correctness under four rules; Rule 1 is the message semantics rule, Rule 2 is the verification rule, Rule 3 is the jurisdiction rule, and Rule 4 is the decomposition rule. We use the following general initial assumptions and Rules under BAN logic:

A. *General initial assumptions*

- $P| \equiv P \xrightarrow{A_{PQ}} Q$: P believes P and Q share the same main key A_{PQ}
- $P| \equiv P \xrightarrow{E_{PQ}} Q$: P believes P and Q share the same encryption key E_{PQ}
- $P| \equiv P \xrightarrow{S_{PQ}} Q$: P believes P and Q share the same session key S_{PQ}
- $P| \equiv M$: P believes M
- $P|\sim M$: P once said M
- $\mathrm{P} \lhd M$: P sees (receives) message M
- $P \mid \Rightarrow M$: P has jurisdiction over M
- $\#(M)$: The formula (M) is fresh
- $\{M\}_k$: The formula $\{M\}_K$ is encrypted under encryption key K
- $\{m\}_H$: The formula $\{m\}_H$ is hashed under the hash function H

B. *Rules*

1. *Message semantics Rule (Rule 1)*

$\frac{P|\equiv Q \xrightarrow{A_{PQ}} P, \mathrm{P} \lhd (\{M\}_k|\{m\}_H)}{P|\equiv Q\ |\sim (\{M\}_k|\{m\}_H)}$: If P believes that P and Q communicate on the shared main key ($P \xrightarrow{A_{PQ}} Q$), then P sees the message $(\{M\}_k|\{m\}_H)$ is encrypted under key K or hash function H or both of them and needs to perform decryption on it, then P believes that Q once said $(\{M\}_k|\{m\}_H)$. By this rule, we ensure that P did not send itself the message under the initial assumption that the message $(\{M\}_k|\{m\}_H)$ is generated by Q and A and P are not equal.

2. *Verification Rule (Rule 2)*

$\frac{P|\equiv \#(M), P|\equiv Q|\sim (\{M\}_k|\{m\}_H)}{P|\equiv Q|\equiv M}$: By Rule 2, if P believes that Q said $(\{M\}_k|\{m\}_H)$, then P has once believed that Q has once said $(\{M\}_k|\{m\}_H)$. By this rule, applying the belief and freshness assertion of saying P believes that the message $(\{M\}_k|\{m\}_H)$is freshly generated, then P is expected to believe that Q believes $(\{M\}_k|\{m\}_H)$. Note that $(\{M\}_k|\{m\}_H)$ can either be encrypted, hashed or both to ensure P of the integrity of $(\{M\}_k|\{m\}_H)$.

3. *Jurisdiction Rule (Rule 3)*

$\frac{P|\equiv Q \Rightarrow (\{M\}_k|\{m\}_H), P|\equiv Q(\{M\}_k|\{m\}_H)}{P|\equiv (\{M\}_k|\{m\}_H)}$: By Rule 3, if P believes that Q has jurisdiction over $(\{M\}_k|\{m\}_H)$, where it is true or not, then if P trusts Q on the truth of $(\{M\}_k|\{m\}_H))$, then P must believe $(\{M\}_k|\{m\}_H)$.

4. Decomposition Rule (Rule 4)

$$(a)\ \frac{P \lhd (M,\mathrm{N})}{P \lhd (M)};\ (b)\ \frac{P|\equiv \#(M)}{P|\equiv \#(M,N)};\ (c)\ \frac{P|\equiv \#(M),\ P|\equiv \#(M),N)}{P|\equiv (M)}$$

In Rule 4, the several rules for the transmitted message can be decomposed and used transmitted message for freshness evaluations or judging. In (*a*), P can see all the parts of the message *M*, *N* if can see all message *M*; in (*b*), the rule states that for any combination of the transmitted message *M* if discovered to be fresh, then the rule operates under the assumption that also one the transmitted message part is fresh; and (*c*) states that if any combination of several messages is believed, then they can be believed individually. To further illustrate the proposed algorithm's logic correctness using BAN logic, we have reconstructed the general initial assumptions and rules following the transmissions of our proposed algorithms based on their notations and steps.

(1) By Rule 1:

$$\frac{EDs| \equiv EDs \xrightarrow{A_{AppKey}} KeySvr,\ \mathrm{EDs} \lhd (\{M\}_{(EDs,KeySvr)}|\{m\}_H)}{EDs| \equiv KeySvr\ |\sim(\{M\}_k|\{m\}_H)}$$

(2) By Rule 2:

$$\frac{EDs| \equiv \#(M),\ EDs| \equiv KeySvr|\sim(\{M\}_k|\{m\}_H)}{EDs| \equiv KeySvr| \equiv M}$$

(3) By Rule 3:

$$\frac{EDs|\equiv KeSvr \Rightarrow(\{M\}_k|\{m\}_H),\ EDs| \equiv KeySvr(\{M\}_k|\{m\}_H)}{EDs| \equiv (\{M\}_k|\{m\}_H)}$$

(4) Applying Rule 4 (*c*):

$$EDs \xrightarrow{A_{AppKey}} KeySvr, \text{ then:}$$

(5) Re-applying Rule 1:

$$\frac{EDs| \equiv EDs \xrightarrow{E_{(\mathrm{NwkS,EDs})}} NwkS,\ \mathrm{EDs} \lhd (\{M\}_{(NwkS,EDs)}|\{m\}_H)}{EDs| \equiv NwkS\ |\sim(\{M\}_{(NwkS,EDs)}|\{m\}_H)}$$

(6) By Rule 2:

$$\frac{EDs| \equiv \#(M),\ EDs| \equiv NwkS|\sim(\{M\}_{(NwkS,EDs)}|\{m\}_H)}{EDs| \equiv NwkS| \equiv M}$$

(7) By Rule 3:

$$\frac{EDs|\equiv NwkS \Rightarrow(\{M\}_{(NwkS,EDs)}|\{m\}_H),\ EDs| \equiv NwkS(\{M\}_{(NwkS,EDs)}|\{m\}_H)}{EDs| \equiv (\{M\}_{(NwkS,EDs)}|\{m\}_H)}$$

(8) Re-applying Rule 4 (*c*):

$$EDs\ | \equiv EDs \xrightarrow{A_{(\mathrm{NwkS,EDs})}} NwkS, \text{ then:}$$

(9) Re-applying Rule 1:

$$\frac{NwkS| \equiv NwkS \xrightarrow{E_{(\mathrm{KeySvr,NwkS})}} KeySvr,\ NwkS \lhd (\{M\}_{(KeySvr,NwkS)}|\{m\}_H)}{NwkS| \equiv KeySvr\ |\sim(\{M\}_{(KeySvr,\mathrm{NwkS})}|\{m\}_H)}$$

(10) Re-applying Rule 2:

$$\frac{NwkS| \equiv \#(M),\ NwkS| \equiv KeySvr|\sim(\{M\}_{(KeySvr,NwkS)}|\{m\}_H)}{NwkS| \equiv KeySvr| \equiv M}$$

(11) Re-applying Rule 3:

$$\frac{NwkS|\equiv keySvr \Rightarrow(\{M\}_{(NwkS,EDs)}|\{m\}_H),\ NwkS| \equiv KeySvr(\{M\}_{(KeySvr,NwkS)}|\{m\}_H)}{NwkS| \equiv (\{M\}_{(KeySvr,NwkS)}|\{m\}_H)}$$

(12) Re-applying Rule 4 (*c*):

$$NwkS \mid \equiv EDs \xrightarrow{E_{(\mathrm{NwkS,EDs})}} NwkS, \text{ then:}$$

(13) By re-applying Rule 1:

$$\frac{NwkS| \equiv EDs \xrightarrow{E_{(,\mathrm{NwkS,EDs})}} NwkS,\ \ NwkS \lhd (\{M\}_{(KeySvr,NwkS),}EDs \xrightarrow{E_{(\mathrm{NwkS,EDs})}} NwkS| \{m\}_H)}{NwkS| \equiv EDs \mid\sim EDs \xrightarrow{E_{(\mathrm{NwkS,EDs})}} NwkS} \quad (1)$$

(14) By re-applying Rule 2:

$$\frac{NwkS| \equiv \#\left(EDs \xrightarrow{E_{(\mathrm{NwkS,EDs})}} NwkS\right), NwkS| \equiv EDs| \equiv\ EDs| \sim (\{M\}_{(KeySvr,EDs)}, EDs \xrightarrow{E_{(\mathrm{NwkS,EDs})}} NwkS| \{m\}_H}{NwkS| \equiv EDs \mid \equiv EDs \xrightarrow{E_{(\mathrm{NwkS,EDs})}} NwkS}$$

(15) Finally, the following are derived: $NwkS \mid \equiv EDs| \equiv EDs \xrightarrow{E_{(\mathrm{NwkS,EDs})}} NwkS$ and $EDs \mid \equiv NwkS \mid \equiv EDs \xrightarrow{E_{(\mathrm{NwkS,EDs})}} NwkS$ to prove that EDs and NwkS can authenticate one another through a trusted third party (KeySvr). In line with the Scyther to prove the security efficiency of our proposed algorithms, BAN logic is applied as the foundation of proving the logical correctness of our proposed algorithm. The discussion results generated by Scyther are discussed as follows.

6. Security Evaluation

This section discusses the security evaluations of the proposed Algorithms A and B. The evaluations are based on security claims and the improved security of trusted key servers. Following the rules of the Scyther tool, Algorithm A was considered secured against all possible LoRaWAN attacks for the Eds, as shown in Table 2, whereas, as shown in Table 3, the NwkS and the KeySvr, are secured against all possible LoRaWAN attacks but only within the bounds. However, a lot of skepticism regarding security level arises as some of the attacks might exist outside the bounds of the whole network which intruders can exploit. Moreover, as depicted in Table 4, Algorithm B was proposed to improve Algorithm A by eliminating the security level sceptics by verifying all security automatic claims for NwkS and KeySvr. Furthermore, our proposed Algorithms A and B were compared to similar models based on the security claims namely secrecy, aliveness, weakagree, nisynch, and niagree. Security claims of Algorithms A and B are discussed as follows.

Table 2. EDs security claims.

Algorithm	Entity	Claims					
	EDs	*Transmission Parameters Secrecy*	*Alive*	*Weakagree*	*Nisynch*	*Niagree*	*Security Status*
A	No attacks	No attacks	No attacks	No attacks	No attacks	No attacks	Verified
B (improved)	No attacks	No attacks	No attacks	No attacks	No attacks	No attacks	Verified

Table 3. KeySvr and NwkS security claims.

Entities	Claims					
	Transmission Parameters Secrecy	*Alive*	*Weakagree*	*Nisynch*	*Niagree*	*Security Status*
KeySvr	No attacks within bounds	No attacks within bounds	No attacks within bounds	No attacks within bounds	No attacks within bounds	Verified
NwkS	No attacks within bounds	No attacks within bounds	No attacks within bounds	No attacks within bounds	No attacks within bounds	Verified

Table 4. KeySvr and NwkS security claims.

Entities	*Transmission Parameters Secrecy*	*Alive*	*Weakagree*	*Nisynch*	*Niagree*	*Overall Security Status*
KeySvr	No attacks	No attacks	No attacks	No attacks	No attacks	Verified
NwkS	No attacks	No attacks	No attacks	No attacks	No attacks	Verified

6.1. Secrecy Claim

As shown in Table 3 for the EDs in Algorithms A and B, all the transmission parameters were kept secret throughout the transmission and reception to and from the EDs. With symmetric encryption and hashing being implemented for dynamic session keys generation and distribution, the proposed Algorithms A and B can dynamically adapt to the generation and distribution of the session keys while preserving the secrecy of all the transmission parameters in the end device. The hashing technique is being implemented for authentication and message integrity when the transmitted messages are received at the receiving entity. The secrecy claim at the Eds site was verified with "no attacks" within and out of bounds transmissions. This ensures that both Algorithms are secured against all possible LoRaWAN attacks on the Eds side. However, as shown in Table 5, Algorithm A has a verified security status; however, the NwkS and the KeySvr are only secured against all possible LoRaWAN attacks only within bounds; this may raise a lot of security sceptics regarding the transmission parameters as they should be always secured against internal attacks within bounds and external attacks outside bounds. These sceptics brought about an improved security level for NwkS and KeySvr by proposing Algorithm B.

Table 5. Comparison of security claims with similar algorithms.

Entities	*Algorithms*	*Transmission Parameters Secrecy*	*Alive*	*Weakagree*	*Nisynch*	*Niagree*	*Overall Security Status*
Eds	A	No attacks	No attacks	No attacks	No attacks	No attacks	Verified
	B	No attacks	No attacks	No attacks	No attacks	No attacks	Verified
	[7]	No attacks within bounds	No attacks within bounds	No attacks within bounds	No attacks within bounds	No attacks within bounds	Verified
KeySvr	A	No attacks within bounds	No attacks within bounds	No attacks within bounds	No attacks within bounds	No attacks within bounds	Verified
	B	No attacks	No attacks	No attacks	No attacks	No attacks	Verified
	[7]	No attacks within bounds	No attacks within bounds	No attacks within bounds	No attacks within bounds	No attacks within bounds	Verified
NwkS	A	No attacks within bounds	No attacks within bounds	No attacks within bounds	No attacks within bounds	No attacks within bounds	Verified
	B	No attacks	No attacks	No attacks	No attacks	No attacks	Verified
	[7]	No attacks within bounds	No attacks within bounds	No attacks within bounds	No attacks within bounds	No attacks within bounds	Verified

6.2. Alive and Weakagree Claim

The alive and weakagree of Algorithms A have been verified, as shown in Tables 2 and 3. For Eds weakagree and alive claim in Algorithms A and B, the Eds are secured against all attacks; this guarantees that the entities encrypt the data transmitted to and from the Eds using the same proposed encryption scheme, and the same encryption scheme is employed by all the entities. Additionally, the weakagree and alive claim for Algorithm A, as shown in Table 3, was verified but the NwkS and KeySvr have security only within the bounds again. This brings about some skepticism regarding the transmission scheme and parameters as they might be exploited using a bit-flipping attack to change the data bits or by a DoS attack to deny the entities the transmission This may

influence affecting the security claim of niagree and nisynch. However, proposing an improved Algorithm B, using Scyther, we verified that the weakagree claim guarantees and ensures that the same encryption scheme has been implemented for NwkS and KeySvr, and the alive claim ensures that all the messages transmitted between the NwkS and KeySvr are encrypted with the same encryption scheme and the same hashing function for authentication and calculations of message integrity.

6.3. Niagree

Based on the nature of the traditional LoRaWAN and our proposed Algorithms A and B, the AppKey is pre-programmed into the Eds, but not pre-shared with the KeySvr or the NwkS as in our proposed algorithms comparing it to the traditional LoRaWAN, where the AppKey is shared prior communication. The AppKey is a very important key as it is used to generate the session keys; however, in our proposed Algorithms A and B, the AppKey is received as a parameter at the KeySvr. This is to ensure that both the Eds and the KeySvr agree on the same AppKey within the transmitted Eds timestamp and only the requesting Eds can transmit the AppKey to the designated KeySvr, and no internal attacks can be experienced before communication by exploiting the KeySvr. However, it is not the case in Algorithm A for KeySvr and NwkS, as shown in Table 3, as only security is experienced within bounds. The NwkS and KeySvr need to agree on the same transmission parameters as per niagree claim; however, the parameters are prone to exploitation outside the bounds during transmission if security is only enforced within bounds. This leads to proposing Algorithm B, as shown in Table 4. By implementing the improved steps as discussed in the improved TKMS algorithm, the Scyther tool verified that Algorithm B is secured against all possible attacks in LoRaWAN.

6.4. Nisynch

As shown in Table 4, we aimed at ensuring that all the participating entities guarantee the current time for entities' synchronization of session keys generation and transmission by using a unique timestamp for each entity, and unique prime numbers which enforce security defenses against replay attacks. Moreover, Algorithm B ensures ineffective de-synchronization attacks on entities as the prime numbers are used instead of the nonces, where these nonces are randomly generated without being secured, whereas the computation of prime numbers is easy but difficult to reverse

7. Discussions

7.1. Comparison of Proposed Algorithms Security Claims with Similar Algorithms

As shown in Algorithm 1 of the proposed Algorithm A, the algorithm was secured against several possible LoRaWAN key attacks; however, it had limited security on the NwkS and the KeySvr as Scyther detected that the algorithm is only secured against the attacks only within the bounds. However, to overcome this security inefficiency experienced by Algorithm A, we modified and improved several steps to produce Algorithm B as illustrated in Algorithm 2. The essence of improving Algorithm A to Algorithm B was to maintain an efficient security level with efficient message integrity and entity authentication. Several security efficiencies are attained by Algorithm B in terms of efficient and secure key generation and distribution, message integrity and entity authentication are discussed as follows:

Timestamps: The use of the timestamp guarantees the receiving entity that the received message is freshly generated from the transmitting entity, and no form of replay attack has been identified.

Prime numbers: Commonly in LoRaWAN, nonces are used to prevent replay attacks; however, these nonces are unsecured pseudorandom numbers. In this paper, we employed the use of the prime numbers instead of the nonces, but they still resemble the same functionalities of nonces; this is for a fact that in cryptography, prime numbers are

easy to compute but difficult to reverse. Based on this, the prime numbers are secured in terms of should the intruders manage to exploit the network or the transmitted message, they will find it difficult to perform replay attacks as they must know the combination of numbers that generated the prime numbers.

Hashing: The hashing technique is employed for calculations of message integrity to avoid bit-flipping attacks, where attackers may alter the bits of the transmitted data. Moreover, hashing ensures entity authentication and session keys are only transmitted via hashing to avoid exposing them in the transmission, where symmetric encryption is used. Lastly, the hashing technique has a shorter computation time for message integrity and authentication as compared to symmetric and asymmetric encryption, where symmetric encryption is only used for transmitting the parameters through a request to join and accept join messages. Moreover, hashing performs encryption through functions, where the inverse of the hashed data is unknown to the public.

AppKey distribution: With the traditional LoRaWAN security models, the AppKey is pre-programmed into the EDs and the NwkS or the KeySvr before communication. AppKey is an important key in LoRaWAN wherein most existing models, it is used to generate AppSKey for securing communication between EDs and AppS, and the NwkSKey for securing communication between EDs and NwkS. The AppKey was also used for securing communication between EDs and NwkS and used to encrypt the initial request to join messages. However, this is not a feasible practical security measure as any of these entities can be exploited to attain AppKey before any communication; should the AppKey be exploited, then the whole transmission is breached. In our proposed algorithms, the AppKey was only pre-programmed into the EDs before communication and distributed to the KeySvr as a parameter secured through symmetric encryption with a timestamp used for freshness. However, the symmetric encryption keys used are created between the communicating entities only, and not overlapped to other entities if the transmitted message is forwarded. The AppKey will be newly generated for every transmission, where the old AppKey cannot be used for current transmissions and is also used to generate NwkSKey and AppSKey from two distinct prime numbers for key independency.

Improved TKMS: The employed improved TKMS ensures secure generation and distribution of the session keys and the parameters used to create them. Moreover, the TKMS facilitates most transmissions in the network to the desired receiving entity; this eases up operational complexities from the NwkS as in most LoRaWAN security models the NwkS performs most of the operations.

7.2. Comparison of Proposed Algorithms with Similar Algorithms

In this section, we discuss the comparison of our proposed algorithms and the similar algorithm in [7] considering the shortfalls of the existing algorithm and the solutions implemented in our algorithms to improve the security level in LoRaWAN. Moreover, the discussion is based on the security claims detected by Scyther, as shown in Tables 5 and 6. Though asymmetric encryption is implemented to generate the session keys to secure the communication between the key server and other servers, asymmetric encryption is still heavy on resources and the resource-constrained nature of LoRaWAN should be taken into consideration when designing security models. We positioned our paper proposing a security model in LoRaWAN by implementing symmetric encryption for initiating the communication and securing the transmission parameters between the entities and using a hash function to ensure authentication and message integrity. Symmetric encryption is low consuming on resources as compared to asymmetric encryption; however, hashing has low computational costs and time when performing authentication and verifying message integrity.

As illustrated in [7], the session key (SK2) is used only once for encrypting a message from the key server to the application server, while the session key (SK1), without being updated, is used for several operations such as encrypting messages between the network server and key server and encrypting new messages between the key server and network

server with the same SK1. Session keys should be updated for every transmission to avoid replay attacks. The nonce is used only to secure a message with AppSKey sent between the key server and the application server. This implementation is infeasible for every transmission as all transmissions are to be always secured. In our proposed algorithms, the implementation of prime numbers guarantees message integrity in the transmission when the request to join the message is initiated by the authenticated end device.

In Table 5, the evaluations are based on the security claims for each entity employed in a specific model, and Table 6 is based on automatic security evaluations which are based on assessing and analyzing the security level of the whole model considering all employed entities at once. The security claims for each entity as illustrated in Table 5 show that the existing algorithm in [7] has limited security for EDs, NwkS and KeySvr as our proposed Algorithm A of only being secured within bounds except for the EDs in Algorithm A which has no attacks at all. Though the overall security is verified for the work in [7] and the security claims are being satisfied, it should be considered for further improvement. We proposed an enhanced Algorithm B, which is an improvement of Algorithm A; this is to ensure that no other attacks in future can be detected outside bounds. Furthermore, as shown in Table 6, for verifying automatic evaluations, the existing model in [7] still maintains verified overall security; however, Scyther still detected no attacks within bounds for the KeySvr. The trusted third party as the main key entity to generate, distribute and update the session keys, should be secured at all costs within and outside bounds to avoid any external attacks that might exploit the network from outside the bounds. In our proposed Algorithm A and the existing algorithm in [7], the majority of the transmissions are composed of the same parameters which are infeasible if the transmissions continue for lengthy periods. However, in our enhanced Algorithm B, randomly changing the transmission parameters for authentication and message integrity calculations guaranteed reinforced security within and outside bounds as is difficult for the attackers to guess the parameters for the next transmission or use parameters from the previous transmission to guess the next transmission.

Table 6. Comparison of proposed algorithms automatic security evaluation with similar algorithms.

Entities	*Algorithms*	*Transmission Parameters Secrecy*	*Alive*	*Weakagree*	*Nisynch*	*Niagree*	*Overall Security Status*
EDs	A	No attacks	No attacks	No attacks	No attacks	No attacks	Verified
	B	No attacks	No attacks	No attacks	No attacks	No attacks	Verified
	[7]	No attacks within bounds	No attacks within bounds	No attacks within bounds	No attacks within bounds	No attacks within bounds	Verified
KeySvr	A	No attacks within bounds	No attacks within bounds	No attacks within bounds	No attacks within bounds	No attacks within bounds	Verified
	B	No attacks	No attacks	No attacks	No attacks	No attacks	Verified
	[7]	No attacks within bounds	No attacks within bounds	No attacks within bounds	No attacks within bounds	No attacks within bounds	Verified
NwkS	A	No attacks within bounds	No attacks within bounds	No attacks within bounds	No attacks within bounds	No attacks within bounds	Verified
	B	No attacks	No attacks	No attacks	No attacks	No attacks	Verified
	[7]	No attacks	No attacks	No attacks	No attacks	No attacks	Verified

8. Conclusions

This paper conducted a comprehensive literature review on LoRaWAN's security models and proposed two LoRaWAN algorithms that serve as a foundation for an improved and secured LoRaWAN model. The algorithms were modelled and verified against all possible LoRaWAN attacks using the Scyther security tool based on formal analysis. Based on this tool, the proposed Algorithm A was verified and exhibited security flexibility amongst all possible LoRaWAN attacks with no attacks detected on Eds.

However, the NwkS and the KeySvr, though secured, have some limitations of detecting attacks only within the bounds which can be exploited out of bounds by attacks such as bit flipping, replay, and DoS attacks. These attacks can intercept transited data out of bounds and alter its bits, exploit the timestamps to avoid freshly generated messages, and deny the services to other entities by exploiting the transmission parameters. Consequently, Algorithm A was enhanced to be more robust against all possible LoRaWAN attacks and supports an efficient trusted key management server. By employing Scyther, the security claims of Algorithm B were verified based on Secrecy to ensure parameters are always secured, aliveness and weakagree to guarantee that all the parameters are encrypted and entities implemented with the same scheme. Moreover, the nisynch claim ensures that all the transmission processes and sessions in the network between the entities adhere to all the security specifications of the proposed protocol and all the participating entities adhere to being synchronized in their current state, while the niagree claim guarantees all the participating entities in the same network adhere to and agree on the same parameters to use for their transmission. We also compared the proposed algorithms with a similar model based on the same security claims, as shown in Table 5, while Table 6 presents the automatic evaluation comparisons which justified the Algorithm B. This shows that it is possible to improve an algorithm such as Algorithm A to ensure that it is not only secured within bounds but also outside bounds. Lastly, we applied the BAN logic formal analysis method to prove the logical correctness of the algorithms cryptographically. The proof shows that the improved Algorithm B is logically correct, and its authenticity is guaranteed.

For future work, we aim to extend our work into finding the trade-offs between security level and energy efficiency by designing an integrated model and carrying out a series of experiments to evaluate the effectiveness and performance.

Author Contributions: Conceptualization, K.N. and B.I.; methodology, B.I.; investigation, K.N.; writing—original draft preparation, K.N.; writing—review and editing, B.I. and A.M.A.-M.; supervision, B.I. and A.M.A.-M.; project administration, N.G.; funding acquisition, A.M.A.-M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: This was supported by FNAS, UDSC, and the Department of Computer Science at the North-West University, Mafikeng campus as well as the Council for Scientific and Industrial Research (CSIR) via the Smart Networks collaboration initiative and IoT-Factory Program (funded by the Department of Science and Innovation (DSI), South Africa).

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

References

1. Choi, J.; Kim, Y. An Improved LEA Block Encryption Algorithm to Prevent Side-Channel Attack in the IoT System. In Proceedings of the 2016 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA), Jeju, Korea, 13–16 December 2016; pp. 1–4.
2. Han, J.; Wang, J. An enhanced key management scheme for LoRaWAN. *Cryptography* **2018**, *2*, 34.
3. Hu, Z. Layered Network Protocols for Secure Communications in the Internet of Things; University of Oregon: Eugene, OR, USA, 2021.
4. Kim, J.; Song, J. A dual key-based activation scheme for secure LoRaWAN. *Wirel. Commun. Mob. Comput.* **2017**, *2017*, 6590713.
5. Mahmood, Z.; Ning, H.; Ghafoor, A. A polynomial subset-based efficient multi-party key management system for lightweight device networks. *Sensors* **2017**, *17*, 670.
6. Na, S.; Hwang, D.; Shin, W.; Kim, K.-H. Scenario and Countermeasure for Replay Attack Using Join Request Messages in LoRaWAN. In Proceedings of the 2017 International Conference on Information Networking (ICOIN), Da Nang, Vietnam, 11–13 January 2017; pp. 718–720.

7. Naoui, S.; Elhdhili, M.E.; Saidane, L.A. Trusted Third Party Based Key Management for Enhancing LoRaWAN security. In Proceedings of the 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA), Hammamet, Tunisia, 30 October–3 November 2017; pp. 1306–1313.
8. Roselin, A.G.; Nanda, P.; Nepal, S. Lightweight Authentication Protocol (LAUP) for 6LoWPAN Wireless Sensor Networks. In Proceedings of the 2017 IEEE Trustcom/BigDataSE/ICESS, Sydney, Australia, 1–4 August 2017; pp. 371–378.
9. Ruotsalainen, H.; Zhang, J.; Grebeniuk, S. Experimental Investigation on Wireless Key Generation for Low-Power Wide-Area Networks. *IEEE Internet Things J.* **2019**, *7*, 1745–1755.
10. KTsai, -L.; Huang, Y.-L.; Leu, F.-Y.; You, I. TTP based high-efficient multi-key exchange protocol. *IEEE Access* **2016**, *4*, 6261–6271.
11. Tsai, K.-L.; Huang, Y.-L.; Leu, F.-Y.; You, I.; Huang, Y.-L.; Tsai, C.-H. AES-128 based secure low power communication for LoRaWAN IoT environments. *IEEE Access* **2018**, *6*, 45325–45334.
12. Yang, X.; Karampatzakis, E.; Doerr, C.; Kuipers, F. Security vulnerabilities in LoRaWAN. In Proceedings of the 2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI), Orlando, FL, USA, 17–20 April 2018; pp. 129–140.
13. Qadir, A.M.; Varol, N. A Review Paper on Cryptography. In Proceedings of the 2019 7th International Symposium on Digital Forensics and Security (ISDFS), Barcelos, Portugal, 10–12 June 2019; pp. 1–6.
14. Hamza, A.; Kumar, B. A Review Paper on DES, AES, RSA Encryption Standards. In Proceedings of the 2020 9th International Conference System Modeling and Advancement in Research Trends (SMART), Moradabad, India, 4–5 December 2020; pp. 333–338.
15. Dhanda, S.S.; Singh, B.; Jindal, P. Lightweight cryptography: A solution to secure IoT. *Wirel. Pers. Commun.* **2020**, *112*, 1947–1980.
16. Zhang, Q. An Overview and Analysis of Hybrid Encryption: The Combination of Symmetric Encryption and Asymmetric Encryption In Proceedings of the 2021 2nd International Conference on Computing and Data Science (CDS), Stanford, CA, USA, 12–17 August 2021; pp. 616–622.
17. Lozupone, V. Analyze encryption and public key infrastructure (PKI). *Int. J. Inf. Manag.* **2018**, *38*, 42–44.
18. Long, S. A Comparative Analysis of the Application of Hashing Encryption Algorithms for MD5, SHA-1, and SHA-512. *J. Phys. Conf. Ser.* **2019**, *1314*, 012210.
19. Zhu, S.; Zhu, C.; Wang, W. A new image encryption algorithm based on chaos and secure hash SHA-256. *Entropy* **2018**, *20*, 716.
20. Zefreh, E.Z. An image encryption scheme based on a hybrid model of DNA computing, chaotic systems and hash functions. *Multimed. Tools Appl.* **2020**, *79*, 24993–25022.
21. Semal, B.; Markantonakis, K.; Akram, R.N. A Certificateless Group Authenticated Key Agreement Protocol for Secure Communication in Untrusted UAV Networks. In Proceedings of the 2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC), London, UK, 23–27 September 2018; pp. 1–8.
22. Cremers, C. "Scyther" Semantics and Verification of Security Protocols. Ph.D. Thesis, University Press Eindhoven, Eindhoven, The Netherlands, 2006.
23. Budiyanto, S.; Santosa, G.B.; Mariati, F.R.I. Upgrading the S-NCI Key Establishment Protocol Scheme to be Secure and Applicable. *IOP Conf. Ser. Mater. Sci. Eng.* **2018**, *453*, 012002.
24. Dalal, N.; Shah, J.; Hisaria, K.; Jinwala, D. A comparative analysis of tools for verification of security protocols. *Int. J. Commun. Netw. Syst. Sci.* **2010**, *3*, 779.
25. Naoui, S.; Elhdhili, M.E.; Saidane, L.A. Enhancing the Security of the IoT LoraWAN Architecture. In Proceedings of the 2016 International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN), Toulouse, France, 26–28 September 2016; pp. 1–7.
26. Lee, J.; Hwang, D.; Park, J.; Kim, K.-H. Risk Analysis and Countermeasure for Bit-Flipping Attack in LoRaWAN. In Proceedings of the 2017 International conference on information networking (ICOIN), Da Nang, Vietnam, 11–13 January 2017; pp. 549–551.