# Performance Evaluation of ESP32 Random Number Generator for LoRa Communication Security


Papat Hidayatulloh
*Research Center for Smart Mechatronics*
*National Research and Innovation Agency (BRIN)*
Bandung, Indonesia
0009-0004-5939-8516

Nada Syifa Qolbiyah
*School of Electrical Engineering*
*Telkom University*
Bandung, Indonesia
nadasyifaqolbiyah@telkomuniversity.ac.id

Asep Nugroho
*Research Center for Smart Mechatronics*
*National Research and Innovation Agency (BRIN)*
Bandung, Indonesia
0000-0001-5021-1154

Husneni Mukhtar
*School of Electrical Engineering*
*Telkom University*
Bandung, Indonesia
husnenimukhtar@telkomuniversity.ac.id

Irwan Purnama
*Research Center for Smart Mechatronics*
*National Research and Innovation Agency (BRIN)*
Bandung, Indonesia
0000-0002-5490-2822

Firman Mangasa Simanjuntak
*School of Electronics and Computer Science*
*University of Southampton*
Southampton, United Kingdom
0000-0002-9508-5849



***Abstract*—We exploit an ESP32-based random number generator (RNG) on LoRa to secure peer-to-peer (P2P) communication. Employing the built-in ESP32 in LoRa could simplify security system designs with no additional RNG hardware required as the security layer. The generated random numbers are a nonce for the initialization vector setup in AES-CTR to enhance the randomness of the ciphertext. It is found that the generated random numbers pass the National Institute of Standards and Technology statistical test suite. This method provides insight into improving the semantic security of the encrypted message by making unique ciphertexts and using AES-Counter modes and AES-Cipher-based message authentication codes as the two layers of LoRa security to check the authenticity and integrity of the message.**



***Keywords—LoRa, ESP32, random number generator, encryption, data security.***


## I. Introduction

In recent years, the application of wireless sensor networks (WSNs) consisting of sensor nodes interconnected by a network has become popular [1] due to its wide implementation in various sectors, such as agriculture, military, healthcare, etc. The Internet of Things (IoT) and Low Power Wide Area Networks (LPWAN) infrastructure development contribute to WSN implementation. One of the most used LPWAN technologies in WSN applications is LoRa.

LoRa, which stands for long range, is a low-powered physical layer of LPWAN technology that employs Chirp-Spread Spectrum modulation [2–6] and operates on unlicensed industrial, scientific and medical bands [2],[4],[7]. Its coverage range is up to 15 km in rural areas and 5 km in urban areas [7],[8]. However, LoRA lacks data transmission security since it uses radio frequency bands accessible to everyone.

Several strategies have been proposed to enhance LoRa security, and most studies implemented the advanced encryption standard (AES) encryption method [9–12] via LoRaWAN (LoRa-based LPWAN protocol) [13–15]. LoRaWAN employs AES-CTR (counter modes) to encrypt the payload and AES-CMAC (cipher-based message authentication code) to ensure packet integrity [13],[14],[16].

Abboud *et al*. implemented and evaluated the AES-256 encryption mode to enhance the security of the LoRaWAN protocol [10]. The implementation of AES-256 provides stronger protection against threats than that of AES-128; however, the transmission time and energy consumption are higher than that of AES-128. The choice between AES-128, and AES-256 depends on the transmitted data. AES-256 is suitable for data transmission that needs superior security, such as healthcare and defense, while AES-128 is less secure but more energy efficient.

Tsai *et al.* proposed the Secure Low Power Communication method, implemented for AES-128 LoRaWAN IoT environments [12]. In this method, the encryption key and lookup table are updated periodically in the end device and server for security enhancement. The AES-128 round process was also reduced to 5 rounds to reduce the computational complexity and save encryption power. However, it is only applied to the application layer, and the key for the MIC code does not update periodically.

The AES method has also been implemented to secure LoRa peer-to-peer (P2P) communication. P2P is beneficial for simple applications that do not need a gateway, making the system easy to build, low cost and efficient [6], [17]. Iqbal *et al*. implemented AES encryption in ESP32 and point-to-point LoRa, employing 64-bit MAC for SCADA applications [18]. Another approach was proposed by Amelia *et al.* [8], which was to implement AES-256 for LoRa-based Asset Tracking. The system was implemented on Arduino Uno and LoRa RFM95. However, this research does not have a message authentication check. Manuel *et al*. proposed LoRa-based secured P2P communications to control and monitor robots [17] by mounting individual LoRa on the robot and using the encrypt-then-MAC (EtM) method to secure the data, which the location information was encrypted with AES-128 and SHA-256 was used for HMAC calculation [17]. However, none of these proposed methods exploits the random number generator (RNG) for securing LoRa communication. RNG is important for encryption in network security [19].

In this work, we increased the security of LoRa communications by adding randomness to encrypt the message, thus providing semantically secure communication. We evaluated the performance of a random number generator (RNG) based on ESP32 for LoRa security. The random number was implemented into 128-bit AES to encrypt the message transmitted over LoRa P2P communication. Moreover, we employed AES-CMAC to authenticate the received message, and the generated random number was used as an AES-CTR initialization vector (IV).

## II. COMPONENTS AND METHODS

To enhance the security of LoRa P2P communication, this study uses an ESP32-based random number generator (RNG) to generate an initialization vector (IV) for AES-CTR message encryption. Additionally, AES-CMAC is implemented to verify the integrity and authenticity of the messages. The system workflow is depicted in the flowchart shown in Fig. 1.

### A. Hardware Setup

Cosmic Lora Aurora board V2 made by Cosmic.id [20], was used in this study. This board is ready to use and contains LoRa and ESP32 modules. This study uses two LoRa Aurora boards as shown in Fig. 2. First LoRa is set as a transmitter and the second board is set as a receiver. Both of LoRa are connected via 433 MHz radio frequency. This frequency band is allowed for LPWAN applications by The Ministry of Communication and Informatics of The Republic of Indonesia [21].

### B. Random Number Generator

ESP32 has a hardware RNG capability that generates random numbers based on physical noise sources [22], hence it can provide true random numbers. To get a physical noise source in the ESP32 RNG, SAR ADC and high-speed ADC are enabled. High-speed ADC is enabled automatically when the Wi-Fi or Bluetooth is activated [22]. Fig. 3 illustrates the block diagram of RNG on ESP 32, which consists of SAR or a high-speed analog-to-digital converter (ADC) to capture the thermal noise. Another noise source is the RC fast clock. Utilizing XOR logic would produce an asynchronous clock mismatch between ADC serial data streams and the RC fast clock, generating a 32-random bit [22].

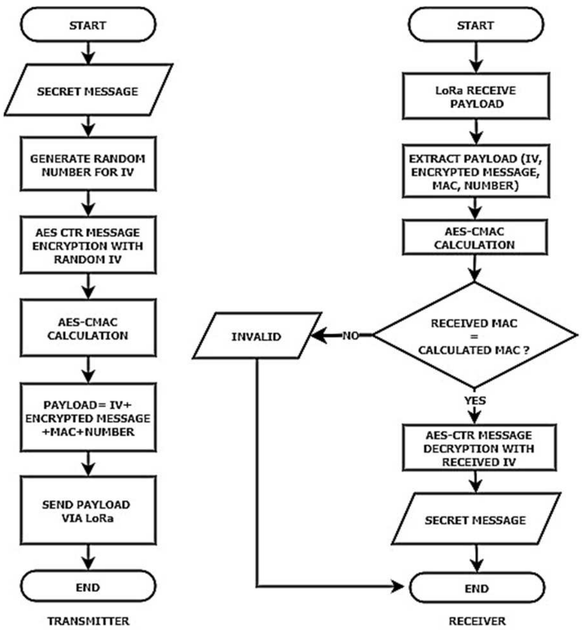


Fig. 1. Flowchart of the system

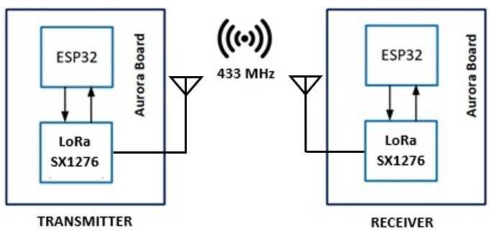


Fig. 2. System setup

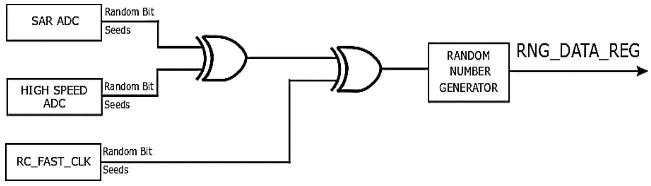


Fig. 3. Block diagram of RNG ESP32 based on physical noise source [22]

The random number generated by ESP32 is up to 32-bit by calling the *esp_random( )* function [22]. The National Institute of Standards and Technology statistical test suite (NIST STS) was used to evaluate the randomness of the generated number. It is a comprehensive set of statistical tests designed to assess the randomness of binary sequences produced by random or pseudorandom number generators [23]. It has become a standardized method for evaluating the quality of random number generators used in cryptographic applications, simulations, and other fields that require high-quality random data. The source code of the NIST STS is open-source, written in C, and can be modified into another programming language. In this study, the NIST STS evaluation was conducted using MATLAB.

The *esp_random( )* function is called multiple times to generate a 96-bit random number. The generated random number was used as an initialization vector (IV) in the AES-CTR encryption process. This number was combined with a 32-bit counter to initialize the 128-bit counter block in the AES-CTR, adding randomness to the encrypted message and increasing security.

### C. Message Encryption and Message Authentication Code

This method implemented 128-bit AES and used two security layers: message encryption using AES-CTR and MAC with AES-CMAC. For this purpose, two secret keys were used: "*keyCTR*" for AES-CTR and "*keyCMAC*" for AES-CMAC. These keys are predefined and used for the entire encryption and decryption process. Message encryption provides data confidentiality, while MAC ensures that the authorized receiver only proceeds with authenticated messages.

#### 1) AES-CTR

The AES-CTR symmetric encryption algorithm was used to encrypt the message. It employs parallel processing during the encryption and decryption process [24]. The steps in the AES-CTR encryption process are initialization setups, block counter generation, counter block encryption, XOR operation and counter block increment [24]. The simplified block diagram of AES-CTR encryption is shown in Fig. 4.

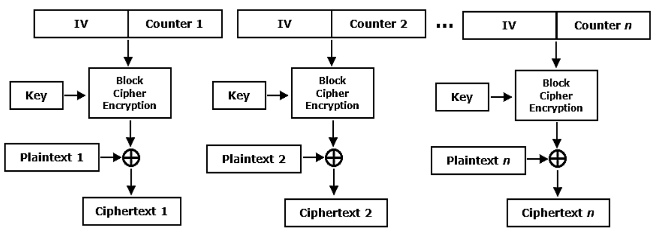


Fig. 4. Block diagram of AES-CTR encryption

The IV was combined with a counter to ensure each block counter was unique. AES algorithm encrypts the counter blocks sequentially, resulting in a stream of pseudo-random key stream blocks. XOR operations were applied between the generated key stream block and plaintext to produce secure ciphertext [25]. To prevent the reuse of the same keystream, the counter block was incremented for each subsequent. The counter in the decryption process was initialized to the same value as in encryption and was incremented for each block to maintain synchronization. Each counter block was processed using the forward cipher function, generating blocks of the same size as the plaintext. These blocks were then XORed with the corresponding ciphertext blocks to recover the original plaintext blocks [24].

Getting unpredictable and unique IV is essential to prevent potential vulnerabilities and ensure data security [24], [26]. Reusing the same IV may enable an attacker to detect patterns within the ciphertext [24]. Hence, ESP32-based RNG was used as a nonce or IV. This IV was utilized in both the encryption and decryption processes. However, this IV did not need to be secret and may be sent with ciphertext [24]. The ciphertext result was encoded into Base64 format before it was transmitted.

#### 2) *AES-CMAC*

The AES-CMAC method was applied to the random number and the encrypted message for message authentication. Both transmitter and receiver calculate the MAC. AES-CMAC provides stronger data integrity than a checksum or an error-detecting code; it is designed to detect intentional and accidental data modifications [27], [28]. CMAC algorithm relies on a symmetric key block cipher. The key used in AES-CMAC should be secret [28]. MAC calculation process using AES-CMAC including initialization, padding, subkey generation, message processing, XOR operations and finalizations [28]. AES-CMAC can be used efficiently over a broad range of message sizes. The data is padded as necessary to align with the block size of the AES algorithm. AES-CMAC does not require an IV during its operation [27]. AES-CMAC generates a MAC by inputting a secret key, a message, and message length in octets [27].

After the MAC calculations process, the hexadecimal MAC value was concatenated with the random number, encrypted message and packet number. The structure of transmitted data is shown in Fig. 5. On the receiver side, payload data were extracted to get the random number, encrypted message, MAC value, and packet number. This information was required to decrypt the message. Before the message decryption process, MAC values were calculated to validate the message. This value is then compared with the received MAC value to check the integrity of the message. If those values were identical, the message was stated as authentic and proceeded to decrypt. Different MAC values made the MAC check fail, and the message will not proceed to decrypt. This ensures that the receiver only processes the message from an authorized transmitter.

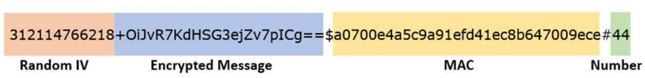


Fig. 5. Payload structure

## III. Results and Discussions

### *A. Random Number Generator Test*

At least 1 million bits of data were generated by ESP32 RNG, and the data passed seven over eight NIST test items, indicating the generated bit stream has sufficient randomness for cryptography operations, as summarized in Table I. In this case, ESP32-based RNG failed the rank test which means it still has a dominant linear dependence between subsequences of the generated numbers. However, it is still acceptable for implementation in cryptography operations.

The generated random number is uniformly distributed, as shown in Fig. 6. This means that each possible outcome is equally likely. Uniform random number distribution ensures unpredictability in cryptographic applications. Additionally, the scatter plot of 1000 RNG data points, shown in Fig. 7, reveals no distinct pattern, indicating that the data is unpredictable and demonstrates good randomness.

The 96-bit fixed-length random number was implemented as an IV in the encryption process. Based on ten times of IV generation tests, the execution time is about 98 µs, as shown in Table II. However, enabling a Radio Frequency (RF) subsystem (Wi-Fi or Bluetooth) to get entropy sources may increase power consumption.

Table I. NIST STS Results

| Item | Result | p-value |
|---|---|---|
| Frequency Test | Pass | 0.856 |
| Block Frequency test | Pass | 0.242 |
| Cumulative Sums test | Pass | 0.957 |
| Runs test | Pass | 0.301 |
| Longest Run of Ones test | Pass | 0.191 |
| Rank test | Not Pass | 0.12e-8 |
| Discrete Fourier Transform test | Pass | 0.238 |
| Non Overlapping Templates test | Pass | 0.524 |

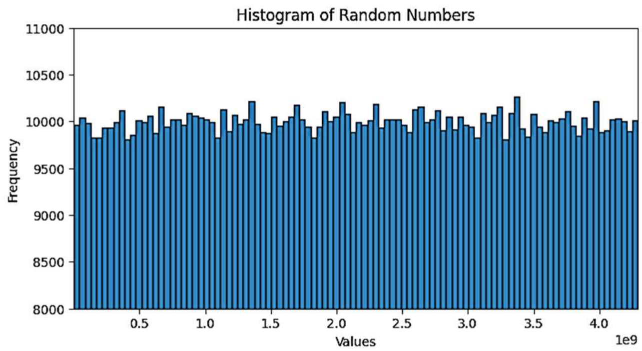


Fig. 6. Histogram of Random Number Distribution

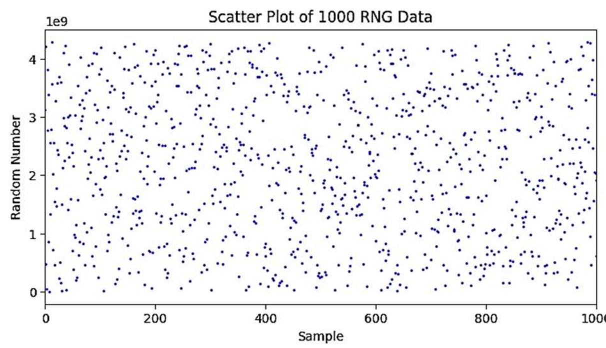


Fig. 7. Scatter plot of 1000 RNG data

TABLE II. EXECUTION TIME

| No. | Execution time (µs) | | | |
|---|---|---|---|---|
| | RNG | Encryption | Decryption | MAC |
| 1 | 98 | 36 | 19 | 44 |
| 2 | 98 | 36 | 19 | 44 |
| 3 | 99 | 37 | 20 | 43 |
| 4 | 99 | 36 | 19 | 43 |
| 5 | 98 | 37 | 19 | 44 |
| 6 | 98 | 37 | 19 | 43 |
| 7 | 99 | 36 | 19 | 44 |
| 8 | 98 | 36 | 20 | 43 |
| 9 | 98 | 36 | 20 | 44 |
| 10 | 99 | 37 | 20 | 42 |

### B. Message Encryption and Message Authentication Code Test

The random numbers were employed as an initialization vector or IV for message encryption to add randomness to the ciphertext output. Without the random IV, Identical plaintext resulted in identical ciphertext, as shown in Fig. 8(a). On the other hand, using the random IV even on the identical plaintext will result in unique ciphertext, as shown in Fig. 8(b). The ciphertext and the key are extremely hard to guess since the ciphertext results are unique. This ensures that no information about the plaintext or key can be obtained by knowing the ciphertext, thus making the encrypted message semantically secure.

Ten times of encryption, decryption and MAC calculation were performed to evaluate the execution time of those processes. This experiment was applied to different plaintext and IV. Each plaintext is 128-bit size. The results are shown in Table II. The average execution time of message encryption is about 36 µs, and the decryption is about 19 µs. Meanwhile, the MAC calculation time is about 43 µs.

### C. System Test

To evaluate the implementation of LoRa security, several experiments were conducted. The experiment was done by several scenarios, which are: *1)* the receiver knows both keys and IV *2)* the *keyCMAC* used by the transmitter and the receiver are different 3*)* the receiver does not know the *keyCTR 4)* the receiver does not know IV. A 10-digit random number was employed as a dummy message for simulation purposes.

Based on the experiment results as shown in Fig. 9–12, AES-CTR and AES-CMAC were successfully implemented for securing LoRa. The receiver checks the message's authenticity and integrity by calculating the MAC. The decryption process is performed only on the authentic message, which means that the message was sent by an authorized transmitter and has never been modified. To decrypt the message, the receiver must know the key and initialization vector (IV). The success of MAC validation and message decryption is shown in Fig. 9.

The wrong *keyCMAC* led to an invalid MAC and failed decryption process as shown in Fig. 10. This indicates that the message was sent by an unauthenticated user. The wrong *keyCTR* also resulted in the wrong decryption as shown in Fig. 11. The decryption result appears random, with some characters being unknown and not displayable on the serial monitor. The output does not correspond to the secret message. Even if the receiver knows the *keyCMAC* and the message validation is successful, the encrypted message cannot be decrypted correctly without *keyCTR*. Hence, the message confidentiality is guaranteed. On the other hand, the wrong IV led to an invalid MAC check and failed message decryption process as shown in Fig. 12. Besides that, the random IV generated by the ESP32 RNG also contributes to LoRa security since it adds the randomness of the ciphertext. This made it extremely hard for unauthorized users to guess the keys and recover the message. It ensures that the only way to decrypt the message is with the correct *keyCTR* and IV.

```
NO      PLAINTEXT          IV                 ENCRYPTED                      MAC
=====================================================================================
1     secret_message#1            qqx0t1I4MH7SXxqSpelDuA==  b7ad064f3d6fa1fa13fe7aae6dd5eb33
2     secret_message#1            qqx0t1I4MH7SXxqSpelDuA==  b7ad064f3d6fa1fa13fe7aae6dd5eb33
3     secret_message#1            qqx0t1I4MH7SXxqSpelDuA==  b7ad064f3d6fa1fa13fe7aae6dd5eb33
```

(a)

```
NO      PLAINTEXT           IV               ENCRYPTED                     MAC
=========================================================================================
1     secret_message#1  315277909342  Aw1YQYLq2Btt3ATkGrmj7w==  264542c58f9c07f10ee99596014c98a6
2     secret_message#1  155877967234  S3kwWyzBsUK1LZj1HpiIDQ==  6c70a87fb417bf90fb3e94a4de5ed30a
3     secret_message#1  345643166530  fLJoNDh0QBjHGAgRyLKmHA==  a9675ffeb7da62aebaff995e9206849b
```

(b)

Fig. 8. Encryption of identical plaintext (a) without random IV (b) with random IV

```
NO    PLAINTEXT        IV              ENCRYPTED                      MAC
==================================================================================
1     1210611766   819126288352  MTvrOa5jDT5+7kDFZGk51Q==   6c06c756b615205b86258956f2cd9615
2     1173460268   247443316731  j/unHojjTVIzuqyRMujzTg==   f188e104b003515a3615875da01ebf84
3     1157241380   575834214176  R0Jz6MwpRJAUvRFvlMOCuQ==   5b2c0970062739ad3897e20a76445097
```

(a)

```
NO          ENCRYPTED               IV                    MAC                     CHECK   DECRYPTED
=====================================================================================================
1    MTvrOa5jDT5+7kDFZGk51Q==  819126288352  6c06c756b615205b86258956f2cd9615  VALID  1210611766
2    j/unHojjTVIzuqyRMujzTg==  247443316731  f188e104b003515a3615875da01ebf84  VALID  1173460268
3    R0Jz6MwpRJAUvRFvlMOCuQ==  575834214176  5b2c0970062739ad3897e20a76445097  VALID  1157241380
```

(b)

Fig. 9. Scenario 1: successful MAC validation and message decryption with correct *keyCMAC*, *keyCTR* and IV (a) transmitter (b) receiver

| NO | PLAINTEXT | IV | ENCRYPTED | MAC |
|---|---|---|---|---|
| 60 | 1351733883 | 408526765637 | F1yJ/hMf3EG8f2bAqBLv4g== | 12d97d488662707f37191d782d08f78f |
| 61 | 1317743172 | 297412270264 | WDuxZQTE9bUfwsvp9VXEHw== | e2a28870b9c084c88516c1d7663414ab |
| 62 | 1222002467 | 894035142251 | OGhKCj/uKxEaFJVmMQG0vw== | 4d03938c84b1d071a861d8e97a8cd360 |

(a)

| NO | ENCRYPTED | IV | MAC | CHECK | DECRYPTED |
|---|---|---|---|---|---|
| 60 | F1yJ/hMf3EG8f2bAqBLv4g== | 408526765637 | 277873bdd1a03da1e1aa45ee48dcb58b | INVALID | FAILED |
| 61 | WDuxZQTE9bUfwsvp9VXEHw== | 297412270264 | 4447912818044c8da1ef5131ee0ee8d0 | INVALID | FAILED |
| 62 | OGhKCj/uKxEaFJVmMQG0vw== | 894035142251 | fce99500225c06835e8a15e0f94f4d8b | INVALID | FAILED |

(b)

Fig. 10. Scenario 2: incorrect *keyCMAC* causes Invalid MAC and failed message decryption (a) transmitter (b) receiver

| NO | PLAINTEXT | IV | ENCRYPTED | MAC |
|---|---|---|---|---|
| 120 | 1391097789 | 273690937741 | L3SYSNDdX65NwIPLyADqCw== | 005f5239e0f2b03be8e144edd4478381 |
| 121 | 1113930517 | 527660586863 | cszomWg15tfn/he2kGDmFw== | 06984e31b65e1b695ee6a75931ef187e |
| 122 | 1253630886 | 348212270022 | 6H+tdSlDVea7OYvYdp41yA== | cf893c365e52d02370da7e15f5e33057 |

(a)

| NO | ENCRYPTED | IV | MAC | CHECK | DECRYPTED |
|---|---|---|---|---|---|
| 120 | L3SYSNDdX65NwIPLyADqCw== | 273690937741 | 005f5239e0f2b03be8e144edd4478381 | VALID | 溈�&i□□□#Wz□□□ |
| 121 | cszomWg15tfn/he2kGDmFw== | 527660586863 | 06984e31b65e1b695ee6a75931ef187e | VALID | �����W&����@� |
| 122 | 6H+tdSlDVea7OYvYdp41yA== | 348212270022 | cf893c365e52d02370da7e15f5e33057 | VALID | ※�2I慱5'����□� |

(b)

Fig. 11. Scenario 3: incorrect *keyCTR* causes incorrect message decryption (a) transmitter (b) receiver

| NO | PLAINTEXT | IV | ENCRYPTED | MAC |
|---|---|---|---|---|
| 140 | 1285010401 | 504463382318 | qL0jWpu08qTwnIIEHrwZVQ== | a78a5b64fbd556618da03e01b82b40d9 |
| 141 | 1323834895 | 415808220527 | 0Ws/O6IcDIWcToUpiLSdZg== | 1fd674ad45f529e594867eb967503f3f |
| 142 | 1011225278 | 120491401333 | YpnO/l1IYR0n+s0ITn3UCQ== | a5ae0d0a9746a3c7495f2751589d183f |

(a)

| NO | ENCRYPTED | IV | MAC | CHECK | DECRYPTED |
|---|---|---|---|---|---|
| 140 | qL0jWpu08qTwnIIEHrwZVQ== | | 46d1ed8e0d7b10658864de6c3174ffdd | INVALID | FAILED |
| 141 | 0Ws/O6IcDIWcToUpiLSdZg== | | 82db7507550fad3737aeb8352e653a2b | INVALID | FAILED |
| 142 | YpnO/l1IYR0n+s0ITn3UCQ== | | 9499d7022814e9146bf4d0b152136dfa | INVALID | FAILED |

(b)

Fig. 12. Scenario 4: incorrect IV causes invalid MAC and failed message decryption (a) transmitter (b) receiver

## IV. Conclusion

We studied random number generator (RNG) based on ESP32 for LoRa security. The NIST STS results show that the random number generated by ESP32 has good randomness and can be utilized in cryptography for securing LoRa communications. Two security layers of LoRa were successfully implemented using AES-CTR for message encryption and AES-CMAC for message authentication code. The generated random number employed as an IV for AES-CTR improves LoRa security by adding randomness to the encrypted message, thus improving the semantic security of the system. This method demonstrated that the built-in ESP32-based RNG for LoRa security implementation is promising. Although the RF subsystem (Wi-Fi or Bluetooth) is not an ideal entropy source, other low-power RNG sources could adopt this method to realize a low-powered security system.


## Acknowledgment

We would like to thank the National Research and Innovation Agency (BRIN) for funding and supporting this research through *Rumah Program Purwarupa Sistem Pendukung Keputusan Berdasarkan Analisis Citra Satelit 2024,* and *Program Visiting Researcher 2024 Batch 2*.


## References


[1] M. Faris, M. N. Mahmud, M. F. M. Salleh, and A. Alnoor, “Wireless sensor network security: A recent review based on state-of-the-art works,” *Int. J. Eng. Bus. Manag.*, vol. 15, p. 184797902311572, Jan. 2023, doi: 10.1177/18479790231157220.

[2] Martin Bor, John Vidler, and Utz Roedig, “LoRa for the Internet of Things,” in *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks*, in EWSN ’16. USA: Junction Publishing, 2016, pp. 361–366.

[3] N. El Rachkidy, A. Guitton, and M. Kaneko, “Collision Resolution Protocol for Delay and Energy Efficient LoRa Networks,” *IEEE Trans. Green Commun. Netw.*, vol. 3, no. 2, pp. 535–551, Jun. 2019, doi: 10.1109/TGCN.2019.2908409.

[4] K. Mekki, E. Bajic, F. Chaxel, and F. Meyer, “A comparative study of LPWAN technologies for large-scale IoT deployment,” *ICT Express*, vol. 5, no. 1, pp. 1–7, Mar. 2019, doi: 10.1016/j.icte.2017.12.005.

[5] C. Zhang, J. Yue, L. Jiao, J. Shi, and S. Wang, “A Novel Physical Layer Encryption Algorithm for LoRa,” *IEEE Commun. Lett.*, vol. 25, no. 8, pp. 2512–2516, Aug. 2021, doi: 10.1109/LCOMM.2021.3078669.

[6] G. Callebaut and L. Van Der Perre, “Characterization of LoRa Point-to-Point Path Loss: Measurement Campaigns and Modeling Considering Censored Data,” *IEEE Internet Things J.*, vol. 7, no. 3, pp. 1910–1918, Mar. 2020, doi: 10.1109/JIOT.2019.2953804.

[7] Z. Sun, H. Yang, K. Liu, Z. Yin, Z. Li, and W. Xu, “Recent Advances in LoRa: A Comprehensive Survey,” *ACM Trans. Sens. Netw.*, vol. 18, no. 4, pp. 1–44, Nov. 2022, doi: 10.1145/3543856.

[8] F. Amelia and M. F. Ramadhani, “LoRa-Based Asset Tracking System with Data Encryption Using AES-256 Algorithm,” in *2022*

*International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET)*, Bandung, Indonesia: IEEE, Dec. 2022, pp. 194–199. doi: 10.1109/ICRAMET56917.2022.9991210.

[9] D. Heeger and J. Plusquellic, "Analysis of IoT Authentication Over LoRa," in *2020 16th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, Marina del Rey, CA, USA: IEEE, May 2020, pp. 458–465. doi: 10.1109/DCOSS49796.2020.00078.

[10] S. Abboud and N. Abdoun, "Enhancing LoRaWAN Security: An Advanced AES-Based Cryptographic Approach," *IEEE Access*, vol. 12, pp. 2589–2606, 2024, doi: 10.1109/ACCESS.2023.3348416.

[11] A. Rizzardi, S. Sicari, and A. Coen-Porisini, "Analysis on functionalities and security features of Internet of Things related protocols," *Wirel. Netw.*, vol. 28, no. 7, pp. 2857–2887, Oct. 2022, doi: 10.1007/s11276-022-02999-7.

[12] 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.

[13] O. Seller, "LoRaWAN Security," *J. ICT Stand.*, Apr. 2021, doi: 10.13052/jicts2245-800X.915.

[14] 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.

[15] J. de Carvalho Silva, J. J. P. C. Rodrigues, A. M. Alberti, P. Solic and A. L. L. Aquino, "LoRaWAN — A low power WAN protocol for Internet of Things: A review and opportunities," *2017 2nd International Multidisciplinary Conference on Computer and Energy Science (SpliTech)*, Split, Croatia, 2017, pp. 1-6.

[16] F. Kuntke, V. Romanenko, S. Linsner, E. Steinbrink, and C. Reuter, "LoRaWAN security issues and mitigation options by the example of agricultural IoT scenarios," *Trans. Emerg. Telecommun. Technol.*, vol. 33, no. 5, p. e4452, May 2022, doi: 10.1002/ett.4452.

[17] M. P. Manuel and K. Daimi, "Implementing cryptography in LoRa based communication devices for unmanned ground vehicle applications," *SN Appl. Sci.*, vol. 3, no. 4, p. 397, Apr. 2021, doi: 10.1007/s42452-021-04377-y.

[18] A. Iqbal and T. Iqbal, "Low-cost and Secure Communication System for Remote Micro-grids using AES Cryptography on ESP32 with LoRa Module," in *2018 IEEE Electrical Power and Energy Conference (EPEC)*, Toronto, ON: IEEE, Oct. 2018, pp. 1–5. doi: 10.1109/EPEC.2018.8598380.

[19] M. Aljohani, I. Ahmad, M. Basheri and M. O. Alassafi, "Performance Analysis of Cryptographic Pseudorandom Number Generators," in *IEEE Access*, vol. 7, pp. 39794-39805, 2019, doi: 10.1109/ACCESS.2019.2907079

[20] "Cosmic LoRa Aurora." Accessed: Jun. 14, 2024. [Online]. Available: https://github.com/cosmic-id/cosmic-lora-aurora

[21] Ministry of Communication and Informatics of The Republic of Indonesia, "Peraturan Menteri Komunikasi Dan Informatika Republik Indonesia Nomor 2 Tahun 2023 Tentang Penggunaan Spektrum Frekuensi Radio Berdasarkan Izin Kelas," p. 17, 2023.

[22] Espressif Systems, "ESP32 Technical Reference Manual," 2024.

[23] Lawrence E. Bassham, Andrew L. Rukhin, Juan Soto, James R. Nechvatal, Miles E. Smid, Elaine B. Barker, Stefan D. Leigh, Mark Levenson, Mark Vangel, David L. Banks, Nathanael Alan Heckert, James F. Dray, and San Vo. 2010. SP 800-22 Rev. 1a. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. Technical Report. National Institute of Standards & Technology, Gaithersburg, MD, USA.

[24] M. Dworkin, "Recommendation for Block Cipher Modes of Operation Methods and Techniques," Dec. 2001.

[25] R. Housley, "Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP)," Jan. 2004.

[26] S. J. H. Pirzada, A. Murtaza, T. Xu, and L. Jianwei, "Initialization Vector Generation for AES-CTR Algorithm to Increase Cipher-text Randomness," in 2019 2nd International Conference on Information Systems and Computer Aided Education (ICISCAE), Dalian, China: IEEE, Sep. 2019, pp. 138–142. doi: 10.1109/ICISCAE48440.2019.221605.

[27] JH. Song, R. Poovendran, J. Lee, and T. Iwata, "The AES-CMAC Algorithm," RFC Editor, RFC4493, Jun. 2006. doi: 10.17487/rfc4493.

[28] M. J. Dworkin, "Recommendation for block cipher modes of operation : the CMAC mode for authentication," National Institute of Standards and Technology, Gaithersburg, MD, NIST SP 800-38b, 2016. doi: 10.6028/NIST.SP.800-38b.