Journal on Sustainable Wireless Systems (ISSN: 2582-3167)
https://irojournals.com/irosws/

A Simulation-Driven BB84 Key Derivation Framework for Authenticated File Transmission over ESP32-Based LoRa Networks

Thilagavathy R.[1], Surya N.[2], Saravanakumar S.[3], Mukeshkiran P.[4]

Students, Department of Electronics and Communication Engineering, K.L.N. College of Engineering, Sivagangai, Tamil Nadu, India.

E-mail: [1]thilagavathimahemp@gmail.com, [2]suryanagendran6@gmail.com, [3]sk7986559@gmail.com, [4]mukeshkiran003@gmail.com

Abstract

The hybrid cryptographic file transfer uses BB84 Quantum Key Distribution (QKD) simulation, SHA-256-based key derivation, AES-256-GCM cryptography, and Long Range (LoRa) radio frequency communication. The BB84 simulation is carried out using IBM's Qiskit with an Aer QASM simulator for quantum key distribution, basis reconciliation, and QBER calculation. This key is then converted to a 256-bit key using the SHA-256 technique and sent to Android phones via QR code scanning. Physical radio frequency communication is achieved using two ESP32 boards and LoRa RA-02 chips operating at 433 MHz. A Flutter application encrypts files using AES-256-GCM mode of operation with a 12-byte initialization vector and a 16-byte authentication tag, meaning each file has a fixed 28-byte overhead due to encryption. Encrypted data is sent in 200-byte blocks, utilizing sequence numbers, acknowledgements, retransmissions, and hardware CRC. Functional testing of message, text, image, and audio files demonstrates successful transfer and decryption. Testing with an incorrect key and altered ciphertext shows proper rejection of data based on authentication.

Keywords: BB84 Quantum Key Distribution, AES-256-GCM, SHA-256 Key Derivation, LoRa Communication, Authenticated Encryption, ESP32-Based Wireless Systems, Quantum-Classical Cryptographic Integration.

DOI: https://doi.org/10.36548/jsws.2026.4.003
Extended version of article received from **6th International Conference on Artificial Intelligence, 6G Communications and Network Technologies 2026**
Received: 10.09.2026, received in revised form: 07.10.2026, accepted: 26.10.2026, published: 11.11.2026
© 2026 Inventive Research Organization. This is an open access article under the Creative Commons Attribution-NonCommercial International (CC BY-NC 4.0) License

1. Introduction

Advancements in quantum computing have sparked fears over the future security of traditional public key cryptographic algorithms. Traditional cryptographic algorithms like RSA and Elliptic Curve Cryptography (ECC) depend on mathematical problems, such as integer factorization and the discrete logarithm problem, which are easily solvable by a quantum computer through an efficient algorithm known as Shor's algorithm [3], [5]. This is the main reason research is being motivated in the field of cryptography to ensure safety in computation under evolving threats, specifically in defense, disaster, and critical infrastructure.

Long-range wireless communication poses additional difficulties in situations where internet connections, cell phone networks, and central network services cannot be used. Situations like disaster areas, remote places, and other locations lacking sufficient infrastructure facilities will require a communication solution that does not depend on existing network infrastructure. In this regard, LoRa communication technology is one option, but wireless connections alone not not offer any form of confidentiality or cryptographic integrity for transmitted files.

The proposed research involves the development of a mixed-mode cryptographical communication system, which utilizes the BB84 quantum key distribution scheme, SHA-256 key generation, AES-256-GCM authenticated encryption, and LoRa-based file transfer. The BB84 algorithm is simulated with IBM Qiskit and Aer QASM simulators, whereas two ESP32 microcontrollers, with LoRa RA-02 modules, serve as the wireless communication medium for interaction between two Android devices. Additionally, the generated key is sent to the mobile applications via QR code scanning, rather than Wi-Fi and LoRa communication methods. Finally, the files are encrypted with AES-256-GCM encryption and transferred in 200-byte portions with acknowledgment and hardware CRC validation.

The main contributions of this work are as follows:

- Integrated simulated BB84 key generation, SHA-256-based key derivation, AES-256-GCM encryption, and LoRa communication into an offline file transfer protocol.
- Implemented QR-code based key transfer for Android apps without transferring the key over Wi-Fi and LoRa connections.

- Designed a wireless file transfer system utilizing LoRa RA-02 modules and ESP32 featuring a 200-bytepayload, acknowledgement, retransmission, and hardware CRC.
- Examined transfers of messages, texts, images, and audio files, considering encryption cost and the rejection of erroneous keys and ciphertext.

2. Related Works

The quantum key distribution (QKD) system has been explored as one of the ways of creating secret keys by applying the principles of quantum communication theory. The BB84 protocol is one such technique that employs two conjugated measurement bases for key creation and detection of any intrusion in the system due to QBER analysis. In their work [2], Adu-Kyere et al. explored the modeling and simulation of the BB84 technique using Python and proved the importance of software implementation in QKD systems study. Also, Begimbayeva and Zhaxalykov [3] considered the BB84, B92, and E91 protocols. Rcently, Escanez-Exposito et al. [10] researched the interactive simulation of QKD schemes and their usage in Wi-Fi networks, pointing out the prospects of combining QKD simulation with classical communication schemes. Manimozhi and Mugelan [1] looked into the generation of AES key based on BB84 sifted keys material with classical encryption mechanisms.

Some researchers have also focused on the combination of QKD with other forms of cryptography such as post-quantum cryptography (PQC) and authenticated encryption. For instance, Ghashghaei et al. [4] considered post-quantum authenticated encryption methods to protect the classical communications that accompany QKD. Ricci et al. [5] researched the use of hybrid keys that involve classical, quantum, and post-quantum cryptography in order to practically address the problem of incorporating multiple methods of key generation. Pirandola et al. [6] gave a comprehensive survey of quantum cryptography developments. The integration of QKD and post-quantum cryptography in a hybrid quantum safe cryptosystem has been researched in experiments by Garms et al. [7], and the use of hybrid post-quantum cryptography and QKD for secure communication has been explored by Aquina et al. [8]. All these references collectively prove the significance of integrating quantum-generated keys with classical or post-quantum cryptography. Nevertheless, their focus lies more in cryptography, key integration, or communication systems rather than an offline file transfer system using low-power embedded

wireless technology.

Communication using LoRa technology represents a good choice for transferring data over long distances in resource-limited IoT environments. However, security demands in LoRa communication are not limited to wireless communication only; they also cover confidentiality, integrity, authentication, and proper key management. LoRaWAN security issues, threats, and solutions were studied by Noura et al. [9], who presented several security aspects related to LPWANs. Consequently, cryptographic security must be included in LoRa communication systems in addition to wireless communication.

Despite progress in BB84 simulation, hybrid quantum and classical cryptographic methods, and LoRa network security has made strides, their combination into a working offline communication system is an implementation challenge that needs to be met. This study solves the problem through the use of a simulated BB84 key exchange method, key derivation using SHA-256, importing the keys using QR code, encryption and decryption using AES-256-GCM, and LoRa file sending via ESP32 microcontrollers. As opposed to solutions focused on studying QKD and hybrid cryptography, our implementation is based on testing end-to-end file transfer of text, image, and audio files using a LoRa network and an Android application.

3. Proposed Methodology

The proposed system consists of two stages: simulation of BB84 key exchange via software and physical file transfer using LoRa technology. In the first stage, the BB84 protocol is simulated using IBM Qiskit with the Aer QASM simulator, after which basis reconciliation and Quantum Bit Error Rate (QBER) assessment take place. The resulting key is then hashed using SHA-256, and a 256-bit key is obtained and converted to Base64 format before being sent to Flutter apps via QR-code scanning. In the second stage, the sender software encrypts the chosen file using the AES-256-GCM cipher algorithm and sends the output to the ESP32 transmitter via Wi-Fi. The ESP32 and LoRa RA-02 wireless transceivers send the encrypted message through the 433 MHz wireless transmission channel. On the receiver side, authentication and decryption of the received file take place, and the decoded message will be shown in Figure 1.

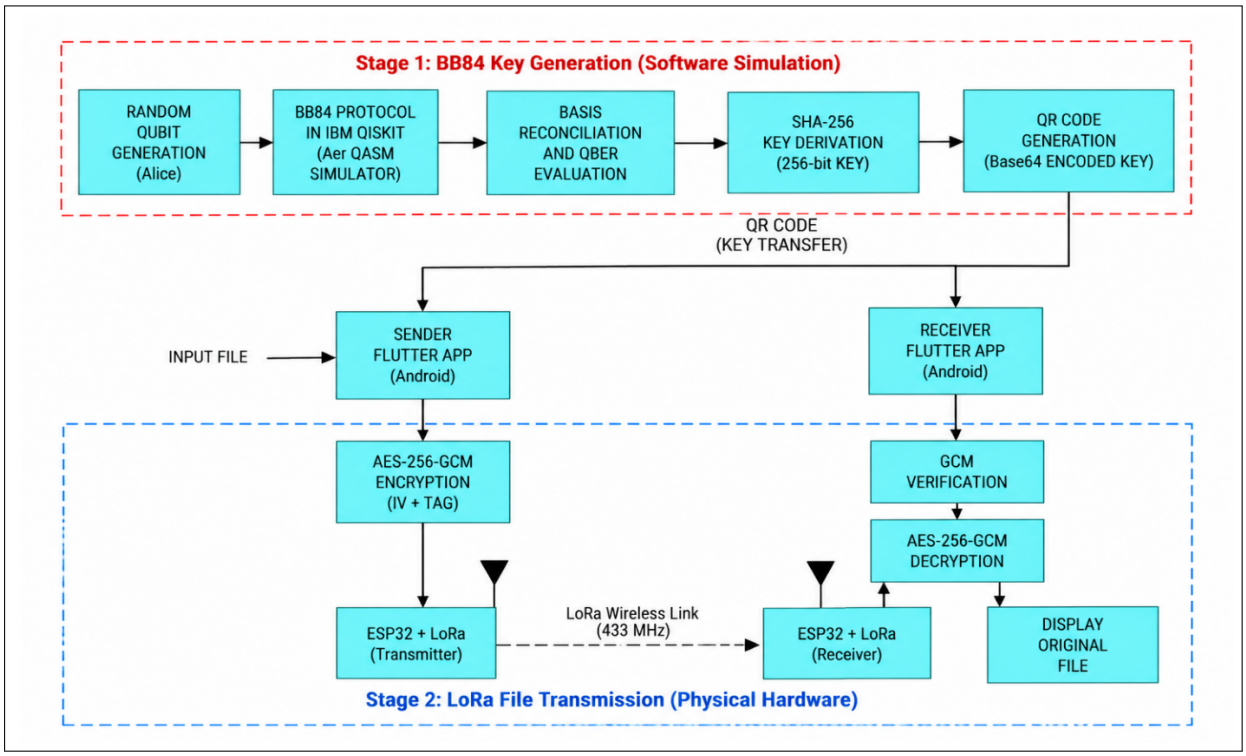


Figure 1. Architecture of the proposed BB84-based secure LoRa communication system

3.1 BB84-Based Key Generation and QBER Evaluation

The BB84 protocol is simulated using IBM Qiskit with the Aer QASM simulator. Alice generates a random bit sequence and selects an encoding basis for each bit from the rectilinear (Z) and diagonal (X) bases. Bob independently selects a measurement basis for each received qubit. After the measurements, the selected bases are compared, and those with different bases are eliminated to get the sifted key.

The Quantum Bit Error Rate (QBER) is calculated by checking a chosen part of the sifted key bits to evaluate the error rate in the simulated BB84 process. An 11% QBER threshold is set as the acceptance criterion for the simulated key generation. This QBER threshold is used as the criterion for determining whether the key generated by the simulation proceeds to the next step of encryption, as it does not indicate the existence or non-existence of a potential attacker. If the QBER threshold is exceeded, the key should be rejected, and the entire key generation process needs to be repeated before moving to the encryption process. However, if the QBER threshold passes the acceptance test, the sifted key is hashed via the SHA-256 algorithm to obtain the 256-bit key to be used for AES-256-GCM encryption. The resulting key is Base64-encoded for

QR-code generation and transfer to mobile applications, as illustrated in Figure 2.

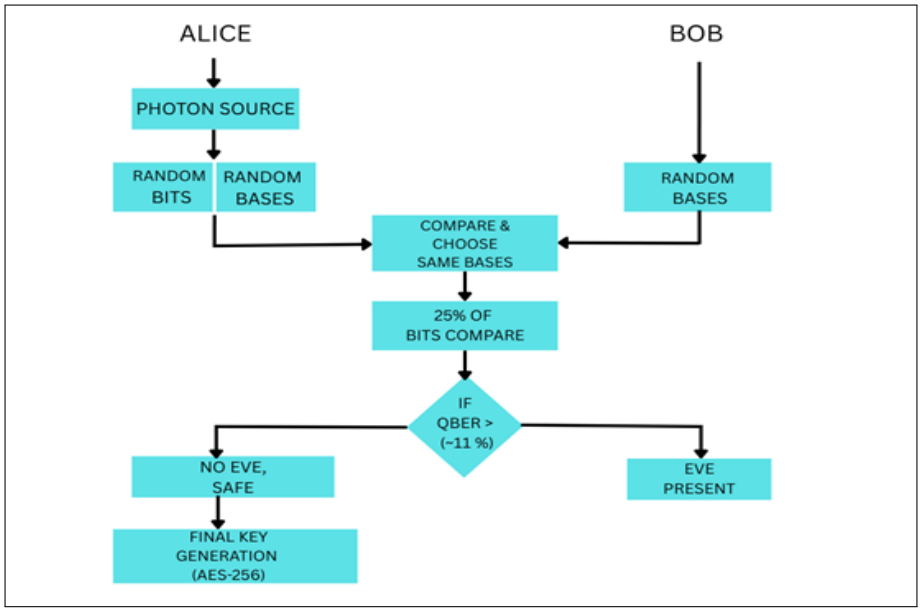


Figure 2. BB84 key generation and QBER-based key acceptance workflow

3.2 Key Distribution Through QR Code

After the QBER-based key acceptance, the generated 256-bits key is encoded in Base64 format and then transformed into a QR code that will be sent to the Android applications developed based on Flutter. The QR code is scanned using the mobile_scanner library to import the shared key to the sender and receiver applications. By doing so, the key will not be transmitted via Wi-Fi or LoRa, and therefore, it will have a lower probability of being intercepted via the wireless transmission channel. Once the key has been imported, the same shared key will be used in both applications to encrypt/decrypt messages using the AES-256-GCM algorithm. As the BB84 protocol is implemented using the Qiskit Aer QASM simulator, the key generation process is only a software-based simulation.

3.3 AES-256-GCM Authenticated Encryption

The shared 256-bit key obtained from the BB84 protocol simulation is utilized to encrypt and decrypt files based on the AES-256-GCM encryption scheme provided by the cryptographic library PointyCastle integrated into the Flutter app. In the process of encryption, a new 12-byte initialization vector (IV) is created individually for each file.. The encryption mode changes the plaintext into ciphertext, while the GHASH function produces the authentication tag in order to secure the encrypted data from any modifications. The encryption output includes the encrypted

data, the IV, and the 16-byte authentication tag, making the encryption overhead constant and equal to 28 bytes per file.

At the receiver, the encrypted payload is processed using the same shared key and the transmitted IV. The authentication tag is verified during decryption before the plaintext is accepted. If verification fails because of an incorrect key or modified ciphertext, the decryption operation is rejected, preventing unauthenticated data from being released. The combined encryption, decryption, and authentication-verification process is illustrated in Figure 3.

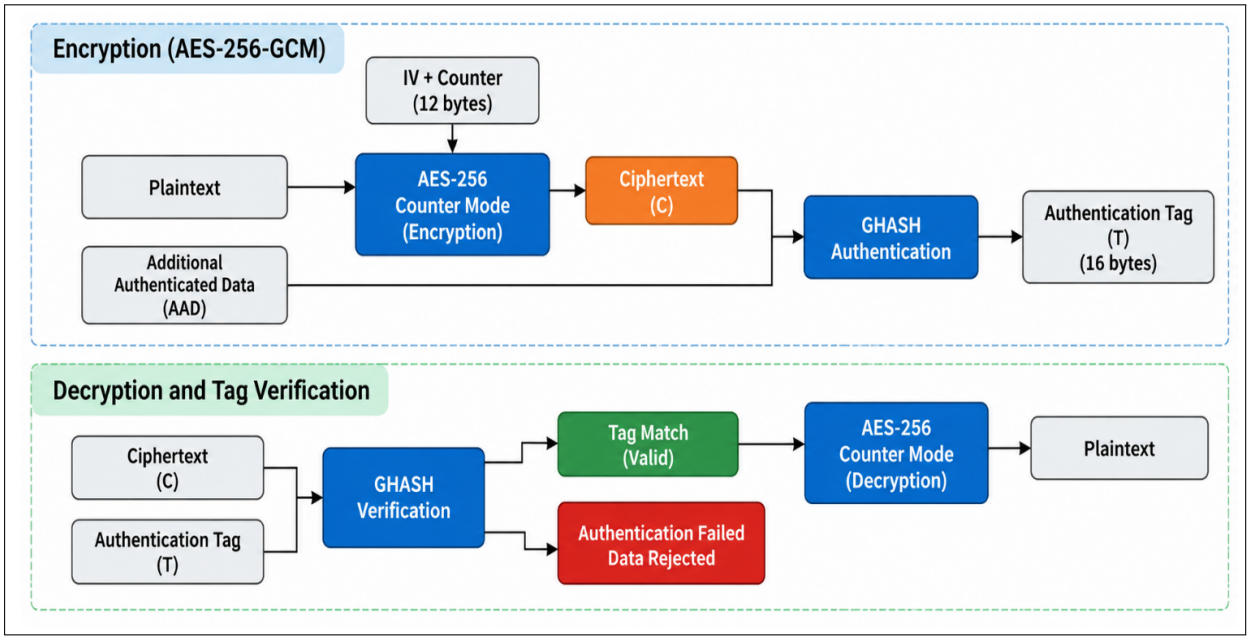


Figure 3. AES-256-GCM encryption, decryption, and authentication verification process

3.4 Hardware and Software Implementation

The proposed architecture will incorporate ESP32 microcontrollers, LoRa RA-02 boards, Android mobile phones, and a mobile app made using Flutter. The BB84 protocol will be emulated using IBM Qiskit and Aer QASM emulator while AES-256-GCM encryption and decryption will be implemented using the PointyCastle library. The ESP32 firmware is developed using the Arduino IDE to support Wi-Fi and LoRa communication.

The hardware elements and their specifications, along with the approximate transmitter-receiver separation distance, antenna setup, test environment, and power source, are presented in Table 1.

Table 1. Hardware components, specifications, and experimental conditions

Component/Parameter	**Specification**
Microcontroller	ESP32 development board
LoRa transceiver	RA-02 LoRa module
Operating frequency	433 MHz
Antenna type	433 MHz LoRa antenna
Transmitter–receiver distance	5 m
Antenna placement and orientation	Approximately 1 m above ground, with vertical antenna orientation
Test environment	Indoor laboratory environment with typical indoor obstructions
Power supply	USB power supply for both ESP32 boards

The software tools, programming environment, and library used for system development are listed in Table 2. These tools enable BB84 simulation, authenticated encryption, QR code scanning, and communication between applications.

Table 2. Software components and libraries

Component	**Technology / Library**	**Purpose**
QKD Simulation	IBM Qiskit (Python), Aer simulator	BB84 simulation, QBER calculation, SHA-256 key derivation
Mobile Application	Flutter (Dart), package name SecureLoRa	Sender/receiver UI, encryption, decryption, file management
Encryption Library	PointyCastle	AES-256-GCM authenticated encryption/decryption
QR Scanning	mobile_scanner	Quantum key import via camera

File Handling	file_picker path_provider, image	File selection, storage paths, automatic image resize to 320x240
Networking	HTTP	HTTP upload/download between app and ESP32
ESP32 Firmware	Arduino IDE 2.x, C++, LoRa.h, WiFi.h, WebServer.h, SPI.h	LoRa TX/RX, WiFi Access Point, HTTP file server

3.5 LoRa-Based Wireless Communication

The wireless communication layer creates the data flow between the transmitter and receiver through the interfacing of an ESP32 microcontroller and a LoRa RA-02 module. Once the file is encrypted, the sender application transfers it to the transmitter ESP32 through local Wi-Fi network through HTTP communication. Firmware splits the encrypted data stream into chunks of 200 bytes for transmission through LoRa. Hardware configuration and communication details of the solution implementation are provided in Table 3.

Table 3. LoRa communication configuration parameters

Parameter	Value
Frequency	433 MHz
Spreading Factor	7
Bandwidth	125 kHz
Coding Rate	4/5
Transmission Power	20 dBm
Chunk Size	200 bytes
Hardware CRC	Enabled

As seen from Figure 4, there is communication between the ESP32 and the LoRa RA-02 via the SPI interface, and as such, the controller is able to manage the entire process of wireless data transmission and reception. The packets are numbered to assist in their identification and

the reconstruction of the file after decryption. Acknowledgements help ensure that the data packets have been received successfully. If the appropriate acknowledgement is not received there is automatic retransmission. At, the receiving end, the ESP32 receives all the data packets and reconstructs the encrypted file. Hardware CRC supports packet-level error detection during transmission.

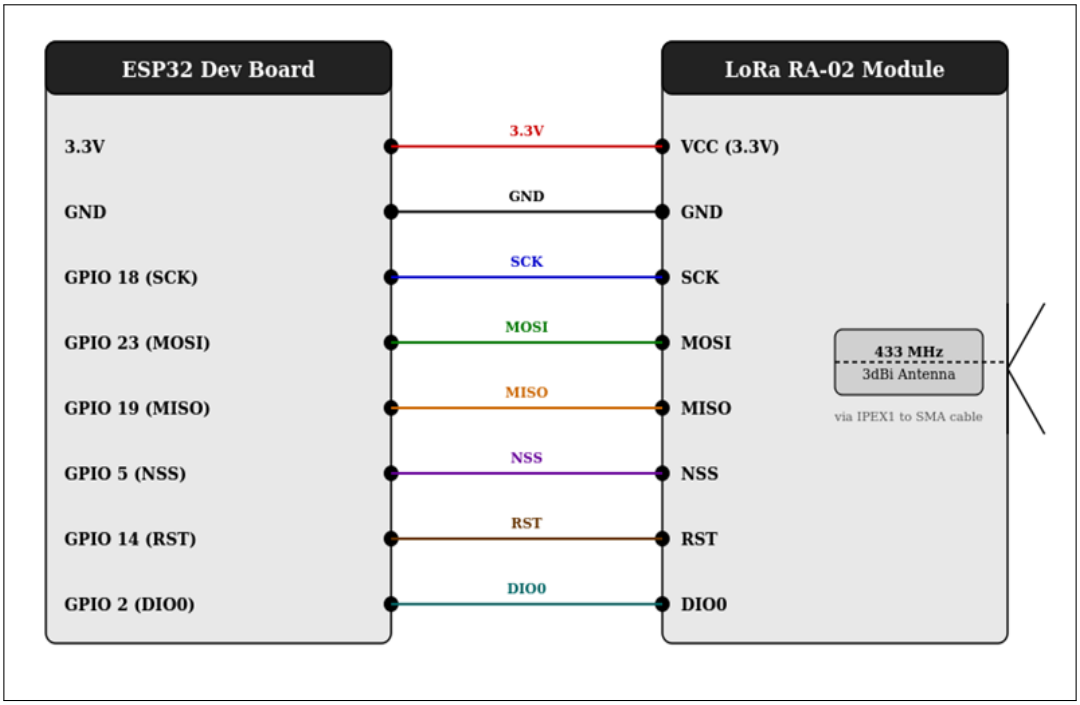


Figure 4. SPI interface connections between ESP32 and LoRa RA-02 module

3.6 Flutter Application and Communication Workflow

Table 4. HTTP communication endpoints between the flutter application and ESP32 firmware

Endpoint	Method	Description
192.168.4.1:80/upload	POST	Upload AES-256-GCM encrypted file to ESP32 transmitter
192.168.4.1:80/status	GET	Poll LoRa transmission progress
192.168.4.1:80/ready	GET	Check whether the reassembled file is ready on the receiver
192.168.4.1:80/download	GET	Download the reassembled encrypted file from the ESP32 receiver

The user interface provided by the Android application developed using Flutter technology

facilitates key import, file encryption, wireless transmission, and receiver decryption. The communication protocol is arranged into sender and receiver actions, where ESP32 firmware acts as the interface between the mobile applications and the LoRa communication channel.

At the sender side, the cryptographic key is retrieved through a QR scan, and the chosen file is encrypted using AES-256-GCM. In the case of image scaling, it is done before the encryption step. The encrypted file is then sent to the transmitter's ESP32 using HTTP requests over local Wi-Fi.. Finally, the ESP32 transmits the payload via the LoRa module to the receiver. HTTP endpoints for file upload, transfer status handling, and file download are provided in Table 4.

At the receiver side, the ESP32 device receives the transmitted information and restores the encrypted file, making it available for the receiving application. The application reads the payload and decrypts and authenticates the file using the secret key. Only the recovered content is considered valid if the AES-GCM authentication is successful. This approach combines the operations of mobile application processes, embedded communication, and file authentication into single process. The application interface and file-transfer workflow are presented in Figure 5.

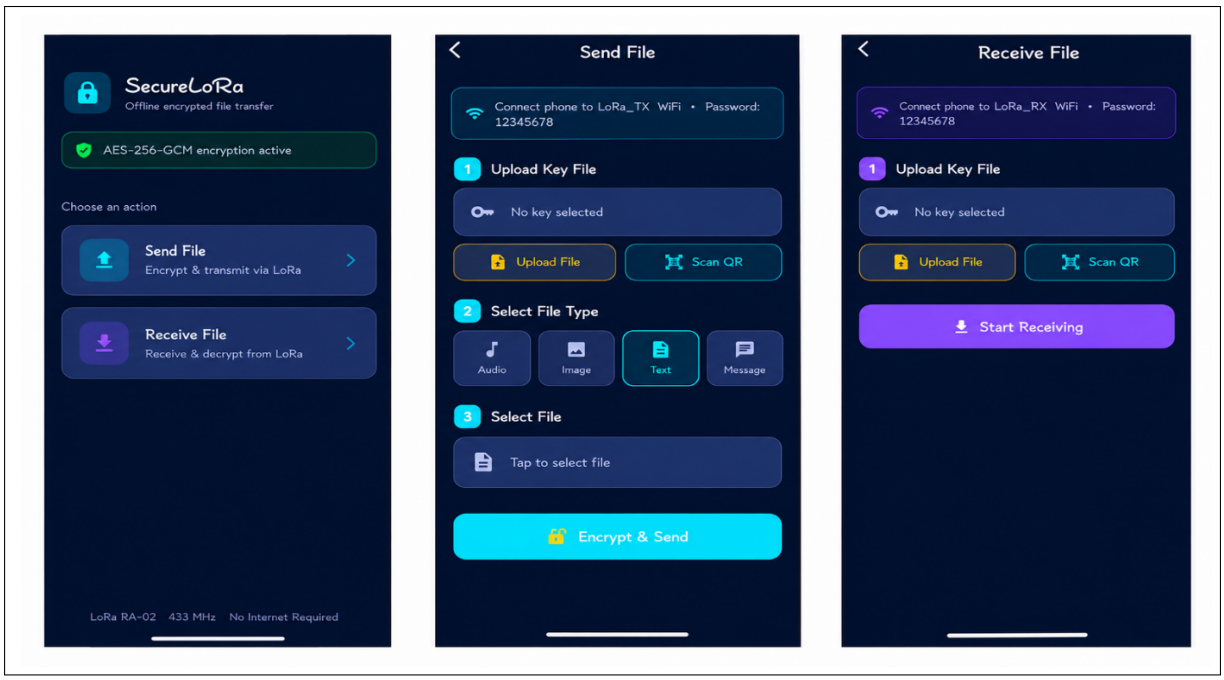


Figure 5. SecureLoRa application interface and file-transfer workflow

4. Results and Discussion

4.1 BB84 Quantum Key Generation Results

The BB84 Key Generation algorithm was implemented using IBM Quantum Qiskit with QASM simulation in the Aer backend. After QBER determination and basis reconciliation, the sifted key was computed through the SHA-256 hash algorithm to obtain an AES 256-bit key, which was then Base64 encoded to be transferred as a QR code to the sender and receiver applications.

As shown in Figure 6, the terminal output displays the simulation results of the BB84 protocol and the key produced by the AES algorithm. The decoded key size was tested to be 32 bytes long, which matches the required size for a 256-bit key for AES-256-GCM encryption. Therefore, the output of the simulated key generation process satisfies the key size requirement for the subsequent encryption process. However, since the BB84 protocol was simulated in a computer simulator, the result demonstrates the simulated key-generation workflow rather than physical quantum key distribution or experimental eavesdropper detection.

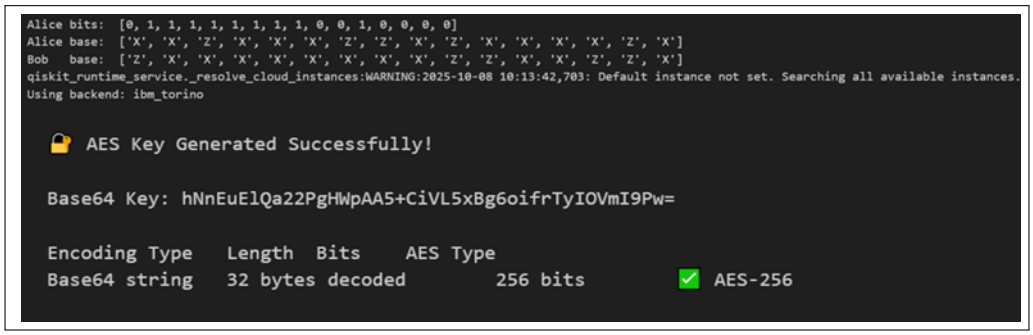


Figure 6. BB84 simulation output showing 256-bit AES key generation

4.2 Hardware Setup

The communication infrastructure involved two ESP32 development boards paired with LoRa RA-02 modules with a transmission frequency of 433 MHz. One node acted as the transmitter while the other acted as the receiver. Data files were sent from the application to the transmitter node via Wi-Fi before encryption and subsequent transmission via LoRa. On the receiver side, received data files were rebuilt and forwarded for authentication and decryption. The hardware evaluation was conducted under a defined transmitter–receiver separation, with the antenna placement and orientation maintained during testing. The surrounding environment and

power-supply configuration were also considered when interpreting the observed file-transmission outcomes.



Figure 7. Assembled ESP32 and LoRa RA-02 transmitter hardware with 433 MHz antenna

The assembled transmitting module is presented in Figure 7, depicting the assembly process of ESP32, LoRa RA-02, and antenna connection. The same type of hardware configuration was employed on the receiving end to facilitate the bidirectional coordination of packet transmission and reception.

4.3 Transmission and Reception Results

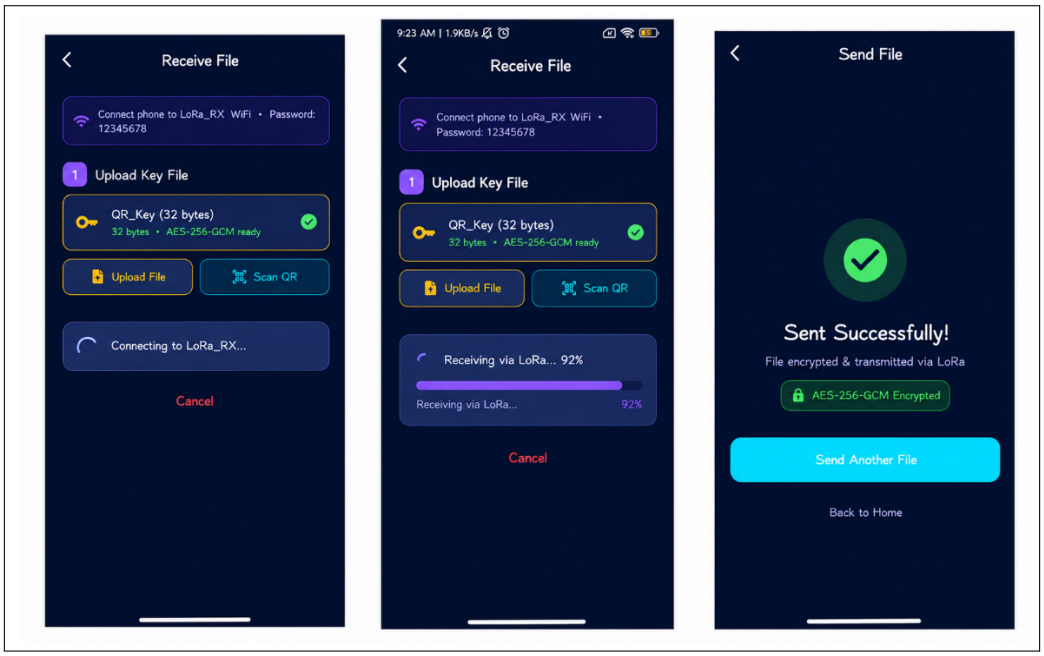


Figure 8. Key import, LoRa reception progress, and transfer confirmation

Information was transmitted from source to destination in an end-to-end manner using four different types of files: a small message, text file, picture, and audio file. In the Flutter app, each file was encrypted using AES-256-GCM, and then the ciphertext was then sent using the LoRa communication channel implemented with ESP32.

Figure 8 demonstrates the process flow, which includes critical imports, receipt processing, and transfer verification.

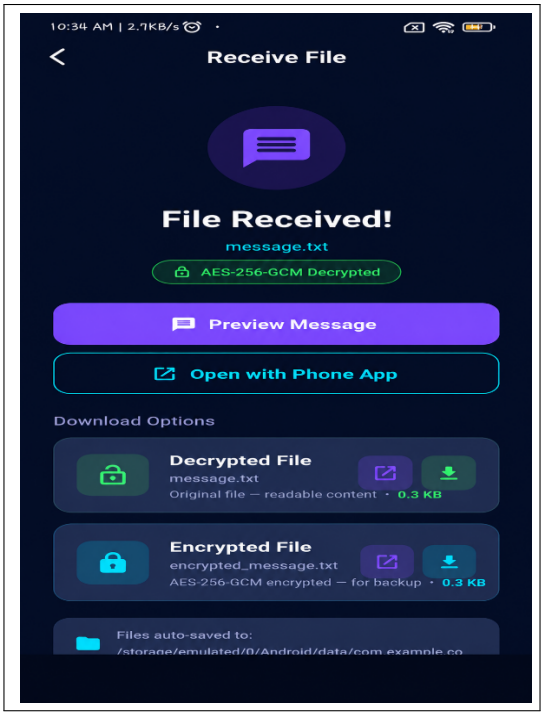


Figure 9. Received and decrypted text message after GCM authentication

Figure 9 shows the received and decrypted text message, illustrating the recovery of the original text at the application level.

Table 5. LoRa transmission and decryption results for different file types

File Type	Encrypted Size	Transmission	Decryption
Message	30 bytes	Success	Success
Text File	1.23 KB	Success	Success
Image (320x240)	3.70 KB	Success	Success

Audio (3 s)	30.03 KB	Success	Success

The transmission outcomes are summarized in Table 5. The tests carried out confirm the successful transmission and decryption of all four file types. This illustrates how the implemented file transfer process functions for all tested payload types.

4.4 Tamper Detection and Encrypted File Verification

The integrity of the file transfer using an encryption process was evaluated through the use of decryption with incorrect keys and modified ciphertext. AES-256-GCM authentication was applied to ensure that data integrity was maintained before data recovery.

These test results show that the decryption process using an incorrect key or modified ciphertext will lead to an InvalidCipherTextException. The authentication process will, therefore, reject such test values without accepting any decrypted output. Figure 10 shows the form of the ciphertext used for file verification.

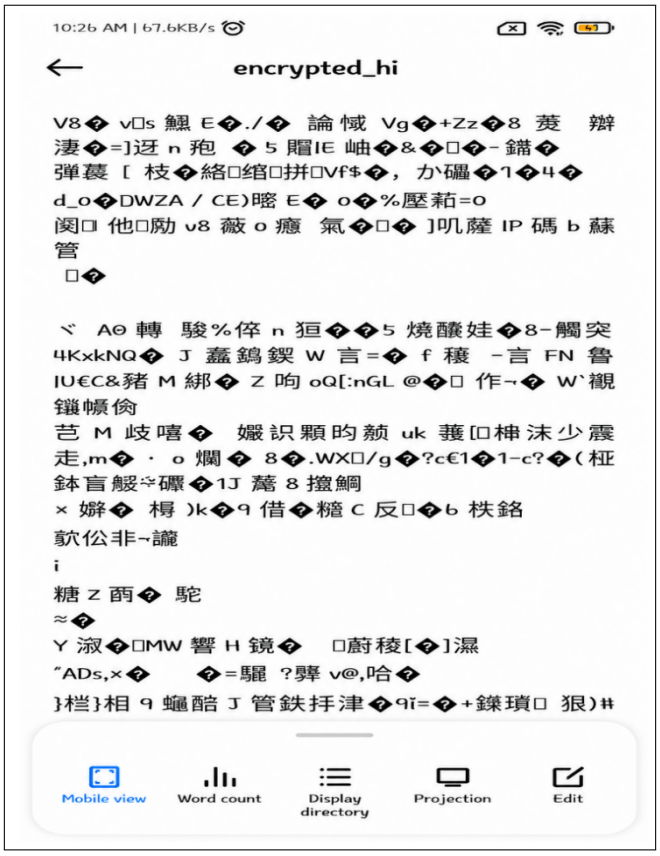


Figure 10. Encrypted ciphertext representation used for file verification

4.5 Encryption Overhead Analysis

The encryption overhead was evaluated by comparing the original file sizes with the corresponding AES-256-GCM encrypted payload sizes. The extra information consistes of 12 bytes for the initialization vector (IV) and 16 bytes for the authentication tag, which leads to a cryptographic overhead of 28 bytes for each encrypted file.

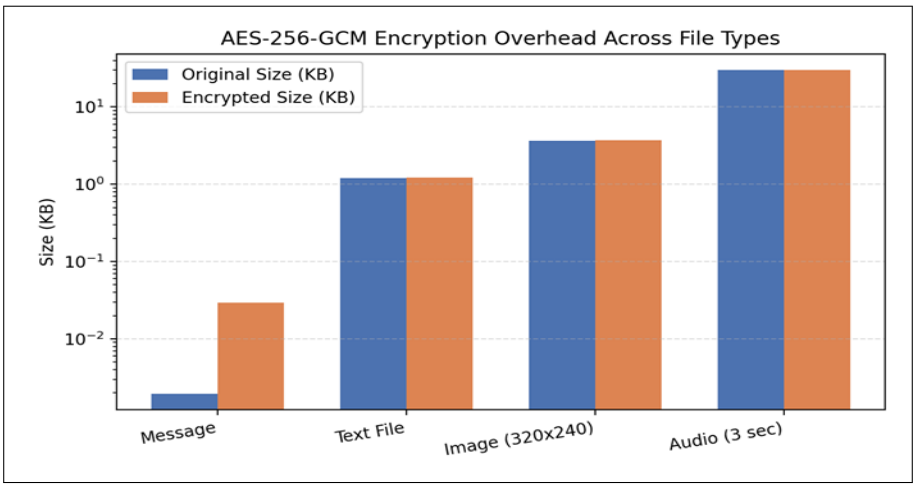


Figure 11. Comparison of original and encrypted file sizes across different file types

Figure 11 represents the comparison between original and encrypted file sizes of the files tested. Fixed overhead is a bigger percentage of the total size of the file when the file is small; however, its relative size reduces as the payload size grows. This means that the cryptographic overhead is constant regardless of the original file size.

4.6 Security Analysis

Security analysis is based on BB84 key generation protocol simulation, AES-256-GCM encryption and decryption, and transmission of an encrypted file through the LoRa communication channel. The BB84 algorithm is performed using IBM Qiskit Aer simulator and the AES-256-GCM protocol is applied to protect the content of the transmitted file.

At the application level, AES-256-GCM provides security and ciphertext integrity using authenticated encryption. The results of the testing process revealed that when decryption was performed using either an incorrect key or altered ciphertext, InvalidCipherTextException was thrown as the tested input was rejected during authenticated decryption. Also, the LoRa communication layer was able to detect errors by employing hardware CRC and acknowledgment

packets.

5. Conclusion

The implemented system comprises BB84 quantum key generation simulation, SHA-256 key generation, AES-256-GCM encryption, and LoRa communication to provide an approach for the safe offline transfer of files. The incorporation of Flutter-based Android applications, ESP32 processors, and LoRa communication allows for transferring messages, text, images, and audio in an encrypted form without reliance on internet access. QR-code key transfer is an additional method for sending the encryption key other than using the wireless communication link. The mentioned experiments show successful file transfer and decryption processes while failed authentication rejects incorrect keys and changed ciphertext inputs. Further work is expected to address physical quantum key distribution, study the performance under different channel scenarios, and measure latency, throughput, packet loss, and transmission range. The integration with post-quantum cryptographic techniques and multi-node LoRa networks will also be researched.

References

[1] Manimozhi, M., and R. K. Mugelan. "Post-quantum AES encryption using ECC points derived from BB84 sifted keys." EPJ Quantum Technology, 2025, vol. 12, no. 1: 109.

[2] Adu-Kyere, Akwasi, Ethiopia Nigussie, and Jouni Isoaho. "Quantum key distribution: Modeling and simulation through bb84 protocol using python3." Sensors, 2022, vol. 22, no. 16: 6284.

[3] Begimbayeva, Y., and T. Zhaxalykov. "Research of quantum key distribution protocols: BB84, B92, E91." Scientific Journal of Astana IT University, 2022, vol. 10: 4-14.

[4] Ghashghaei, Farshad Rahimi, Yussuf Ahmed, Nebrase Elmrabit, and Mehdi Yousefi. "Enhancing the security of classical communication with post-quantum authenticated-encryption schemes for the quantum key distribution." Computers, 2024, vol. 13, no. 7: 163.

[5] Ricci, Sara, Patrik Dobias, Lukas Malina, Jan Hajny, and Petr Jedlicka. "Hybrid keys in practice: Combining classical, quantum and post-quantum cryptography." IEEE Access,

2024, vol. 12: 23206-23219.

[6] Pirandola, Stefano, Ulrik L. Andersen, Leonardo Banchi, Mario Berta, Darius Bunandar, Roger Colbeck, Dirk Englund et al. "Advances in quantum cryptography." Advances in Optics and Photonics, 2020, vol. 12, no. 4: 1012-1236.

[7] Garms, Lydia, Taofiq K. Paraïso, Neil Hanley, Ayesha Khalid, Ciara Rafferty, James Grant, James Newman, Andrew J. Shields, Carlos Cid, and Maire O'Neill. "Experimental Integration of Quantum Key Distribution and Post-Quantum Cryptography in a Hybrid Quantum-Safe Cryptosystem." Advanced Quantum Technologies, 2024, vol. 7, no. 4: 2300304.

[8] Aquina, Nick, Simon Rommel, and Idelfonso Tafur Monroy. "Quantum secure communication using hybrid post-quantum cryptography and quantum key distribution." In 24th International Conference on Transparent Optical Networks (ICTON), IEEE, 2024, 1-4.

[9] Noura, Hassan, Tarif Hatoum, Ola Salman, Jean-Paul Yaacoub, and Ali Chehab. "LoRaWAN security survey: Issues, threats and possible mitigation techniques." Internet of Things, 2020, vol. 12: 100303.

[10] Escanez-Exposito, Daniel, Pino Caballero-Gil, and Francisco Martín-Fernández. "Interactive simulation of quantum key distribution protocols and application in Wi-Fi networks." Wireless Networks, 2023, vol. 29, no. 8: 3781-3792.