COMP-SCI 5577
Internet of Things
Mini-Project2 Report

Professor: Dr Qiuye He
Term members:
Hui Jin
Jiarui Zhu
Harshitha Bashyam
Feb 22, 2025
GitHub Link: https://github.com/nanxuanhui/Iot_miniproject.git
Demo Link: https://drive.google.com/drive/folders/1_SbPbScpDSIjz5L0x5VPETcOR-Ps2eR3?usp=sharing

Content

Introduction

The goal of this project is to design and implement a data acquisition and secure transmission system based on ESP32 and DHT11 sensors, protect data security through AES-256 encryption, and upload encrypted data to a remote Flask server using HTTP protocol. The server decrypts and parses the data to ensure data integrity and security.

In Internet of Things (IoT) applications, data security is critical, especially during remote transmission, sensitive data is vulnerable to network attacks. The core features of this system are:

- Collect environmental temperature and humidity data through DHT11 sensor.
- Use AES-256 encryption to ensure that data will not be interpreted by unauthorized third parties.
- Transmit data through WiFi network and HTTP protocol to improve the reliability of data transmission.
- The server receives, decrypts and processes data based on the Flask framework, which can be expanded to database storage and visual analysis in the future.

Implementation

1. Hardware Setup

The hardware part mainly includes ESP32 development board, DHT11 sensor and WIFI network, among which ESP32 is used as the data processing and communication core, DHT11 is responsible for environmental data collection, and WIFI network is used for data transmission.

- **ESP32 Board**

ESP32 is a low-power WiFi and Bluetooth dual-mode chip that supports high-performance computing and wireless communication. Its main tasks include:

1. Connecting to the WiFi network and communicating with the server.
2. Collecting temperature and humidity data, pre-processing and formatting locally.
3. Using AES-256 for encryption to ensure the security of data transmission.
4. Using HTTP to send data to ensure that the remote server can receive and parse the data.

- **DHT11 Sensor**

The DHT11 sensor is a low-cost temperature and humidity detection module. Its main features include:

1. Temperature detection range: 0°C ~ 50°C, accuracy ±2°C.
2. Humidity detection range: 20% ~ 90% RH, accuracy ±5% RH.
3. Digital signal output, easy for ESP32 to process and read.

- **WiFi Network**

The LED is connected to GPIO2. The LED provides visual feedback based on the sensor data and threshold settings.

2. Software Architecture

The software part mainly includes the ESP32 side and the Flask server side (Python), where ESP32 is responsible for data collection, encryption and sending, and the server side is responsible for data reception, decryption and storage.

- **ESP32**:
  - **WIFI connection**: Connect to the network through WiFi.begin(ssid, password) and check the network status in the loop() loop to ensure automatic reconnection after disconnection.
  - **Get NTP time**: Get time from the network time server through configTime(gmtOffset_sec, daylightOffset_sec, ntpServer) to avoid inaccurate data timestamps caused by local time drift.
  - **DHT11 sensor data reading**: Read temperature and humidity data through dht.readTemperature() and dht.readHumidity(), and perform data verification to ensure data validity.
  - **AES-256 encryption**: AES-256 ECB mode is used to encrypt JSON data, and Base64 encoding is used to facilitate network transmission. Since AES requires 16-byte alignment, PKCS7 padding is used on the ESP32 side to ensure data integrity.
  - **HTTP data sending**: Use HTTPClient to send encrypted data to the Flask server. The server returns the HTTP status code, and the ESP32 end parses and prints the result returned by the server.
- **Flask Server**:
  - Listening for HTTP POST requests: The server listens to the /post-data endpoint for Base64 encrypted data sent by the ESP32.
  - AES-256 Decryption: Use AES-256 ECB mode for decryption and remove PKCS7 padding to ensure correct data parsing.
  - Parsing JSON data: The server-side json.loads() parses the decrypted JSON data and checks the data integrity.
  - Data storage: Currently the server only prints data.

Challenges Faced

1. **AES-256 encryption and decryption failure**:
   During the initial test, the AES encrypted data on the ESP32 side could not be correctly decrypted on the server side, resulting in an AES Decryption Error.
   - Cause analysis:
   - The encrypted data on the ESP32 side was not padded to 16 bytes, resulting in server-side decryption failure.
   - The Flask server side did not correctly remove the PKCS7 padding, resulting in a JSON parsing error.
   - Solution:
   - Add PKCS7 padding on the ESP32 side to ensure that the data block length complies with the AES specification.
   - Remove the PKCS7 padding characters on the Flask side to ensure that the data format is correct after decryption.versa).
2. **Unstable WIFI connection**:

The ESP32 WiFi connection is easily disconnected in some cases, resulting in data being unable to be successfully uploaded.

- Solution:
  - Add WiFi connection detection in loop(), and automatically reconnect when WiFi is disconnected.
  - Turn off ESP32 power saving mode WiFi.setSleep(false) to improve WiFi connection stability.

3. **JSON parsing failed**:
   When the server-side json.loads() parses JSON data, an Extra data error is reported, resulting in data loss.
   - Solution:
     - Use decrypted.strip() on the Flask side to remove extra characters.
     - Add log printing on the ESP32 side to ensure that the data format is correct before encryption.with Serial communication required careful design.

Conclusion

This project successfully demonstrates the integration of an ESP32 board with a DHT11 sensor for real-time environmental data collection and secure transmission. By leveraging AES-256 encryption, the system ensures that temperature and humidity data remain protected during transmission to a remote Flask server. The implementation of a structured workflow—from data acquisition and encryption on the ESP32 to decryption and JSON parsing on the server—illustrates a practical approach to secure IoT communication.

Through careful design and implementation, the system reads sensor data, encrypts it using AES-256 with PKCS7 padding, and transmits the encrypted payload via an HTTP POST request. The Flask server decodes and decrypts the received data, extracts meaningful information, and validates its structure before processing. The use of an NTP server for time synchronization guarantees that timestamps associated with sensor readings remain accurate, preventing issues related to data misalignment or incorrect time logging.

Despite its successful implementation, the project encountered several challenges that required thorough troubleshooting. Ensuring that encrypted data remained properly formatted and aligned for AES decryption was a key focus, as improper padding led to initial decryption failures. Additionally, handling network instability required implementing an automatic WIFI reconnect mechanism to maintain consistent data transmission. Another challenge involved preventing erroneous JSON parsing due to invisible padding artifacts in the decrypted text, which was addressed through careful string manipulation and validation techniques.

Future enhancements to this system could significantly extend its capabilities. One major improvement would be integrating higher-precision sensors, such as the DHT22 or SHT31, to enhance data accuracy and resolution. Adding wireless connectivity options, such as MQTT over WIFI or Bluetooth Low Energy (BLE), would allow real-time data streaming to mobile devices or cloud platforms, making the system more accessible. The user experience could also be improved by

incorporating a web-based dashboard or mobile application, providing graphical visualization of environmental conditions rather than relying solely on serial print outputs.
These improvements would not only enhance the project's functionality but also expand its applicability to more complex and demanding scenarios, reinforcing the importance of secure, real-time data collection and transmission in IoT systems.

Reference

[1] https://lastminuteengineers.com/esp32-pinout-reference/