CHAPTER 1

INTRODUCTION

The Internet of Things, or IOT, refers to the billions of physical devices around the world that are now connected to the internet, collecting and sharing data. Thanks to cheap processors and wireless networks, it's possible to turn anything, from a pill to an airplane to a self-driving car into part of the IOT. This adds a level of digital intelligence to devices that would be otherwise dumb, enabling them to communicate real-time data without a human being involved, effectively merging the digital and physical worlds.

1.1 WHAT IS IOT

A light bulb that can be switched on using a smartphone app is an IOT device, as is a motion sensor or a smart thermostat in your office or a connected streetlight. An IOT device could be as fluffy as a child's toy or as serious as a driverless truck, or as complicated as a jet engine that's now filled with thousands of sensors collecting and transmitting data back to make sure it is operating efficiently. At an even bigger scale, smart cities projects are filling entire regions with sensors to help us understand and control the environment.

The term IOT is mainly used for devices that wouldn't usually be generally expected to have an internet connection, and that can communicate with the network independently of human action. For this reason, a PC isn't generally considered an IOT device and neither is a smartphone -- even though the latter is crammed with sensors. A smart watch or a fitness band or other wearable device might be counted as an IOT device, however.

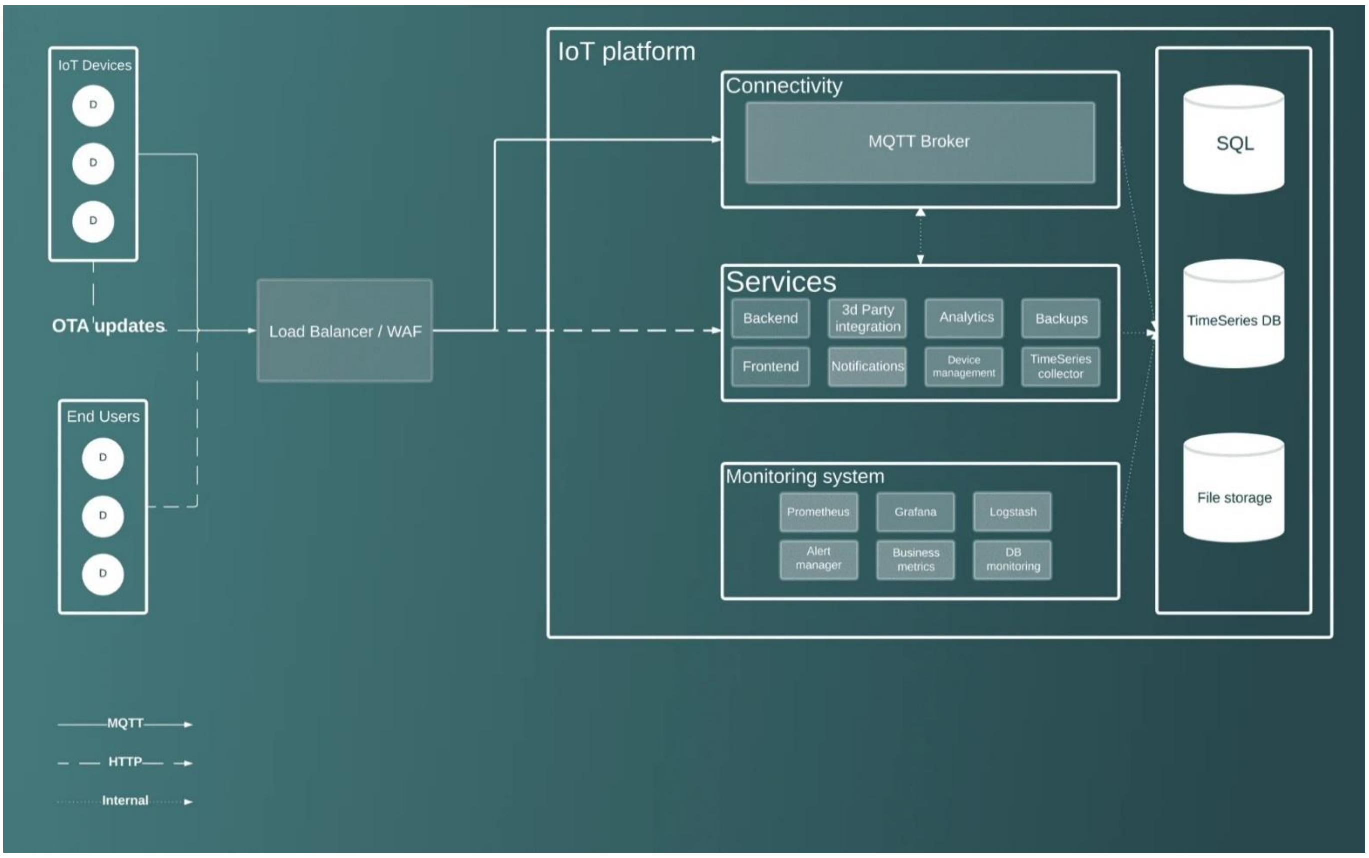


Fig-1.1 – Architecture of IOT

1.2 USES OF IOT

IOT (Internet of Things) is the network of physical objects-devices, vehicles, buildings and other items embedded with electronics, software, sensors, and network connectivity-that enables these objects to collect and exchange data. The IOT concept has faced prominent criticism, especially in regards to privacy and security concerns related to these devices and their intention of pervasive presence. Embedded with electronics, Internet connectivity, and other forms of hardware, these devices can communicate and interact with others over the Internet, and they can be remote monitored and controlled. IOT devices can be used to enable remote health monitoring and emergency notification systems. These health monitoring devices can range from blood pressure and heart rate monitors to advanced devices capable of monitoring specialized implants, such as pacemakers, Fitbit electronic wristbands, or advanced hearing aids.

1.3 ESP-32

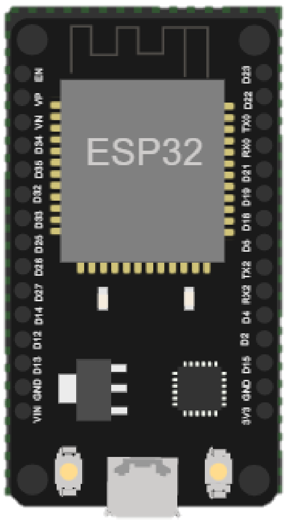


Fig-1.2 – ESP32

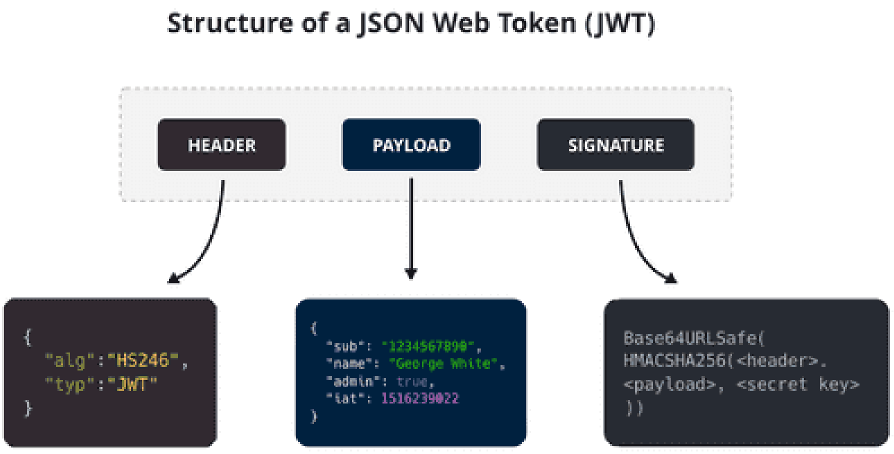
Espressif Systems created the ESP32, a system-on-a-chip (SoC) microcontroller that is inexpensive and low-power. Due to its incorporated Wi-Fi and Bluetooth capabilities, it is frequently utilised for Internet of Things (IoT) applications. A dual-core processor, up to 520KB of SRAM, and a number of interfaces, including UART, SPI, I2C, and ADC, are available in the ESP32. Additionally, it supports a variety of operating systems, including FreeRTOS, making it simple to create and deploy programmes on the platform. The ESP32 is widely utilised in many different applications, including as smart lighting, wearable electronics, and home automation.

The ESP32's integrated Bluetooth and Wi-Fi functions are among its most notable qualities. It is crucial for many IoT applications that connecting to the internet and other devices be simple. An ESP32-based smart thermostat, for instance, may quickly connect to a Wi-Fi network at home and interact with a smartphone app, enabling users to control their thermostat from a distance.

Additionally, the ESP32 supports a number of operating systems, such as FreeRTOS, making it simple to create and deploy applications on the platform.

Popular real-time operating system FreeRTOS offers a straightforward and adaptable method for controlling numerous operations on a microcontroller. Since many IoT applications require complicated apps, this implies that developers may create them with ease.

1.4 JSON WEB TOKENS



```
{
  "alg":"HS246",
  "typ":"JWT"
}
```

```
{
  "sub": "1234567890",
  "name": "George White",
  "admin": true,
  "iat": 1516239022
}
```

```
Base64URLSafe(
HMACSHA256(<header>.
<payload>, <secret key>
))
```

Fig-1.3 – JWT

JWT, or JSON Web Token, is a secure, compact, URL-safe way of representing claims and information between two parties. It’s commonly used to facilitate secure data transfer over the internet and for authentication purposes in web applications. A JWT contains three parts: a header, a payload, and a signature, which are used together to validate the integrity and authenticity of the token.

For example, in a web application, after logging in, a server can generate a JWT and send it to the user’s browser. This token can then be stored by the client and sent with every request to verify that the user is authenticated, enabling secure access without needing to resend credentials. Many modern applications use JWT for managing sessions, as it enables stateless, decentralized, and secure communication.

1.5 Advanced Encryption Standard(AES)

AES (Advanced Encryption Standard) is a symmetric encryption algorithm widely used to secure data. It was established by the National Institute of Standards and Technology (NIST) in 2001 as the replacement for the older DES (Data Encryption Standard). AES is known for its efficiency, security, and versatility across various applications, such as securing sensitive data in financial, governmental, and personal contexts.

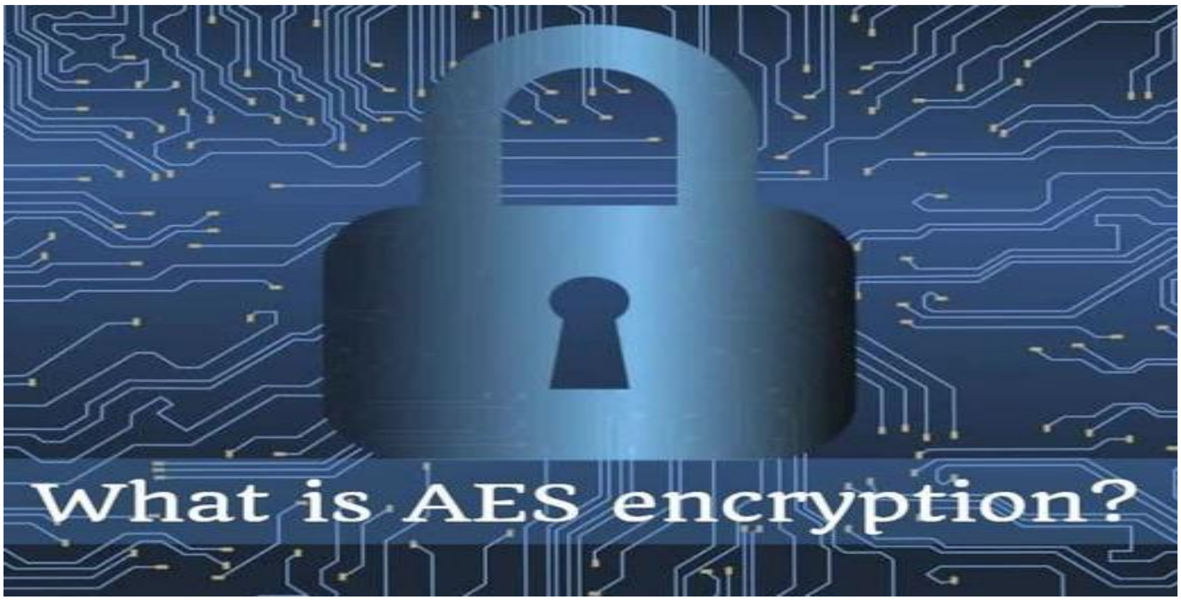


Fig-1.4 - AES

Key Features of AES

1. **Symmetric Key Algorithm**: AES uses the same key for both encryption and decryption, meaning the sender and receiver must share the key securely.
2. **Block Cipher**: AES encrypts data in fixed-size blocks of 128 bits. For messages longer than 128 bits, it operates in modes that divide data into blocks.
3. **Key Sizes**: AES supports three key lengths: 128, 192, and 256 bits. The longer the key, the stronger the encryption (but also the more processing required).
4. **Rounds of Encryption**: Depending on the key size, AES performs multiple rounds of transformation:
    - **AES-128**: 10 rounds
    - **AES-192**: 12 rounds
    - **AES-256**: 14 rounds

Steps in AES Encryption

AES encryption transforms plaintext data using a series of operations:

1. **Key Expansion**: The original encryption key is expanded to create multiple round keys.
2. **Initial Round**:
    - **AddRoundKey**: The initial round key is combined with the plaintext using bitwise XOR.
3. **Main Rounds** (repeated for each round):
    - **SubBytes**: Each byte in the block is substituted with a byte from a fixed substitution table (S-Box).
    - **ShiftRows**: Rows of the block are shifted cyclically to introduce diffusion.
    - **MixColumns**: Columns are mixed to blend the data more completely.
    - **AddRoundKey**: The round key is added to the block.
4. **Final Round** (similar to main rounds but without MixColumns):
    - **SubBytes, ShiftRows, AddRoundKey**: These final transformations generate the ciphertext.

AES Modes of Operation

AES is often used with modes of operation to handle data larger than 128-bit blocks. Common modes include:

- **ECB (Electronic Codebook)**: Each block is encrypted independently, making it fast but less secure for long data.
- **CBC (Cipher Block Chaining)**: Each block is XORed with the previous block's ciphertext, providing better security.
- **CFB (Cipher Feedback) and OFB (Output Feedback)**: These turn AES into a stream cipher, useful for real-time encryption.
- **GCM (Galois/Counter Mode)**: Adds authentication to the encryption, verifying data integrity and authenticity.

Security of AES

AES is considered very secure and resistant to all known practical attacks. The larger key sizes, especially AES-256, offer robust protection against brute-force attacks due to the vast number of possible key combinations.

CHAPTER 2
LITERATURE REVIEW

1**"A Lightweight Security Scheme for IoT Communication Using ESP32" Journal**

Summary: In this study, the authors propose a lightweight security scheme designed for IoT communication that utilizes the ESP32 microcontroller. The scheme incorporates various cryptographic techniques to ensure data integrity and confidentiality without sacrificing performance. The results demonstrate that the proposed scheme is suitable for resource-constrained environments, making it ideal for many IoT applications, including smart cities and industrial automation.

2. "Security Enhancement in IoT Devices Using ESP32"

This study discusses various security challenges faced by IoT devices, particularly those using the ESP32 microcontroller. It highlights the vulnerabilities in standard communication protocols and proposes a framework for enhancing security through advanced encryption standards (AES) and secure socket layer (SSL) protocols. The authors implement a secure data transmission method using the ESP32, demonstrating that the integration of these security measures significantly reduces the risk of unauthorized access and data breaches in IoT applications.

3. "Review of Prevention schemes for Replay Attack in Vehicular Ad hoc Networks (VANETs) "

Vehicular Ad hoc Networks (VANETs) enable automatic message exchange among vehicles to improve traffic management but face security risks due to open communication channels. These networks are vulnerable to security attacks like replay attacks, which researchers have tried to mitigate, though existing solutions often incur high computational costs. This study analyzes replay attack impacts in VANETs and surveys current prevention strategies.

4. "Detecting and Preventing Beacon Replay Attacks in Receiver-Initiated MAC Protocols for Energy Efficient WSNs"

It discusses a significant security concern in Wireless Sensor Networks (WSNs), particularly focusing on the beacon replay attack. To counter this vulnerability, the authors propose a new challenge-response authentication protocol called RAP (Receiver Authentication Protocol). This protocol aims to authenticate the identity of the receiver when initiating communication, thereby preventing unauthorized replay of beacons. The effectiveness of RAP is verified using automated tools like OFMC and ProVerif.

5. "Intrusion Detection System to Overcome a Novel Form of Replay Attack (Data Replay) in Wireless Sensor Networks"

The paper highlights the growing use of Wireless Sensor Networks (WSNs) and their benefits, such as low cost and easy deployment. However, it notes vulnerabilities due to limited resources and wireless communication. Replay attacks, where an adversary intercepts and retransmits messages, can result in false alerts. A more sophisticated variant, the data replay attack, involves replaying only the data field of a message while updating other fields, making detection more difficult. To address this, the authors propose the Data Replay Intrusion Detection System (DR-IDS), which utilizes Anomaly Detection, Signature-Based Detection, and Real-Time Monitoring to identify and mitigate data replay attacks in WSNs.

6. "Implementation of a WiFi-based V2V-V2I Communication Unit for Low Speed Vehicles"

The authors present a prototype communication unit using WiFi technology to enable V2V and V2I communications for low-speed vehicles in urban environments. They stress the importance of real-time data processing for vehicles to respond quickly to changing road conditions, thereby enhancing safety and efficiency. This research supports the vision of smart cities, where connected vehicles interact seamlessly with urban infrastructure to improve mobility and safety.

7. "Design of Roadsite Unit (RSU) Based On ESP32 for Vehicle to Infrastructure (V2I) Communication System. "

The article discusses the rising vehicle numbers at Politeknik Negeri Medan, leading to traffic congestion and parking issues. It proposes a Vehicle to Infrastructure (V2I) communication system for real-time information exchange between vehicles and roadside units (RSUs). The system includes an Onboard Unit (OBU), a Roadside Unit (RSU) built with an ESP32 microcontroller for Wi-Fi and Bluetooth connectivity, and Transmission Channels. The research finds that effective communication occurs at a distance of 90 meters, with a received power level of -91 dBm, which is suitable for the ESP32's sensitivity range.

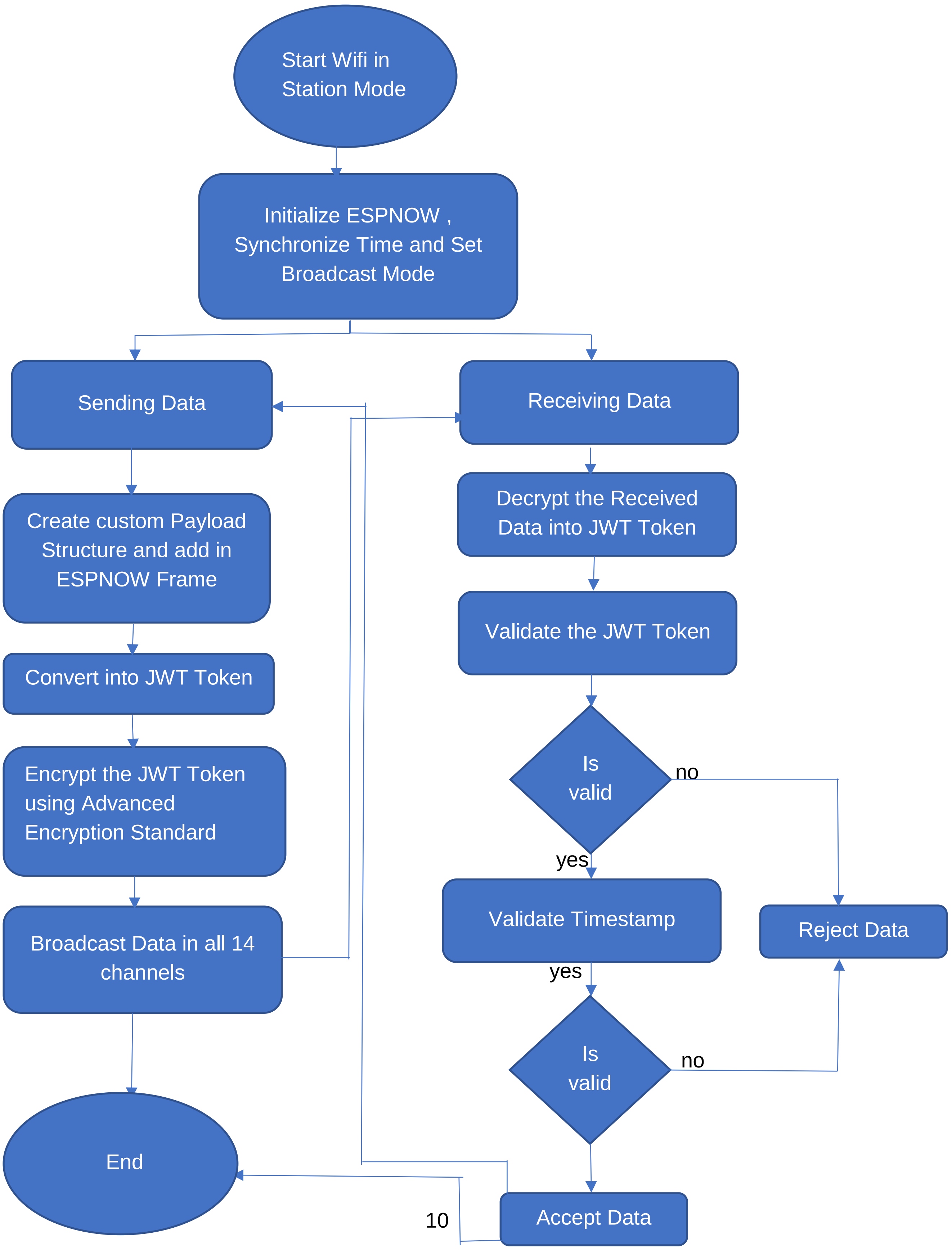
8. "V2X Communication for Message Transmission and Warning Detection"

The paper explores advancements in Vehicle-to-Everything (V2X) communication technologies aimed at improving road safety and traffic efficiency. It highlights the significance of V2X systems, which enable vehicles to communicate with each other and infrastructure, potentially reducing accidents by up to 13% annually. The article outlines two main types of V2X technologies: WLAN-based (802.11p) and Cellular-based (C-V2X). It also addresses challenges in implementing V2X systems, such as the need for reliable communication standards, security against hacking, and managing electromagnetic interference in modern vehicles.

CHAPTER 3

SYSTEM DESIGN

3.1 ARCHITECTURE



Start Wifi in Station Mode
Initialize ESPNOW , Synchronize Time and Set Broadcast Mode
Sending Data
Receiving Data
Create custom Payload Structure and add in ESPNOW Frame
Decrypt the Received Data into JWT Token
Convert into JWT Token
Validate the JWT Token
Encrypt the JWT Token using Advanced Encryption Standard
Is valid
no
yes
Broadcast Data in all 14 channels
Validate Timestamp
Reject Data
yes
Is valid
no
End
Accept Data

3.2 MODULES DESCRIPTION

1. System Initialization Module

- **Description**: This module sets up the ESP32 microcontroller to enable ESP-NOW communication in station mode. It configures network interfaces and initializes required libraries for secure communication.
- **Key Components**:
    - **ESP-NOW Setup**: Configures the ESP32 to operate with ESP-NOW in station mode, enabling peer-to-peer communication.
    - **Channel Settings**: Sets the Wi-Fi channels for multi-channel communication.
    - **Error Handling**: Retries setup if initialization fails.

2. Authentication Module (JWT Generation)

- **Description**: This module generates JSON Web Tokens (JWTs) for authenticating messages between devices, ensuring only trusted devices participate in the communication.
- **Key Components**:
    - **JWT Token Creation**: Generates a token with essential data (e.g., device ID, message ID) for verifying message origin.
    - **Validation Mechanism**: Ensures each token is unique to each message, preventing replay attacks.
    - **Error Handling**: Logs and retries if token generation fails.

3. Payload Management Module

- **Description**: This module structures the data to be transmitted, creating a standardized payload that includes device identifiers, message details, and status information.

- **Key Components**:
    - **Data Packing**: Encodes data into a fixed format for consistent transmission, including device ID, message ID, timestamp, data length, and status code.
    - **Payload Size Check**: Ensures that data size stays within defined limits to avoid transmission issues.
    - **Expiration Timestamp**: Adds a timestamp to each message for future validation and time-sensitive data checks.

4. Encryption Module (AES Encryption)

- **Description**: This module applies AES encryption to secure the payload, ensuring that transmitted data is only readable by authorized devices.
- **Key Components**:
    - **AES Encryption**: Encrypts the structured payload to prevent unauthorized access.
    - **Encryption Key Management**: Manages AES keys to ensure secure, consistent encryption.
    - **Error Handling**: Retries encryption if errors occur, logs issues for debugging.

5. Multi-Channel Transmission Module

- **Description**: This module controls data transmission across multiple Wi-Fi channels (1-14) to improve reliability by avoiding congestion and interference.
- **Key Components**:
    - **Channel Switching**: Loops through channels, switching for each data packet to ensure delivery even in congested networks.

    - **Transmission Control**: Sends encrypted payloads to a broadcast MAC address to reach all devices on the network.
    - **Delay Management**: Introduces small delays between channel switches to ensure smooth data flow.

6. Reception and Listening Module

- **Description**: This module is responsible for listening on each channel and receiving messages from other devices.
- **Key Components**:
    - **Channel Monitoring**: Loops through channels 1-14, listening for incoming messages on each.
    - **Message Reception**: Captures incoming encrypted data for further decryption and validation.
    - **Timeouts and Retries**: Implements short delays to manage intermittent packet loss and improve reliability.

7. Decryption and Authentication Module

- **Description**: This module decrypts the received payload using AES and validates the JWT token to authenticate the message origin.
- **Key Components**:
    - **AES Decryption**: Decrypts received data using the AES key shared across devices.
    - **JWT Validation**: Confirms message authenticity by validating the JWT; discards messages with invalid tokens.
    - **Error Logging**: Logs decryption and validation errors for security auditing and debugging.

8. Data Decoding and Validation Module

- **Description**: After successful decryption and authentication, this module decodes the payload and verifies the data’s integrity, checking parameters like timestamp validity.
- **Key Components**:
    - **Payload Decoding**: Extracts data fields (device ID, message ID, etc.) from the decrypted payload.
    - **Timestamp Verification**: Checks if the timestamp is within an allowed range (e.g., within 3 minutes) to prevent replay attacks.
    - **Data Integrity Checks**: Confirms that decoded data matches expected formats and lengths.

CHAPTER 4

PROJECT REQUIREMENTS

4.1 PLANNING OF PROJECT WORK

Date	Progress Milestone	Objective	Completion
July 10	Initiated project concept; chose ESP32 with ESP-NOW protocol for secure multi-channel IoT communication.	Define secure multi-channel IoT communication setup	5%
July 18	Conducted research on security measures; selected AES encryption and JWT authentication to ensure secure data exchange.	Establish security foundation	10%
August 9	Set up initial communication in Wokwi simulator; tested ESP32 devices with basic ESP-NOW data exchange without encryption.	Confirm ESP32-ESP-NOW initial communication feasibility	15%
August 22	Configured Wi-Fi channels; began optimizing multi-channel ESP-NOW communication to reduce interference and enhance reliability.	Improve communication reliability across channels	30%
September 7	Implemented AES encryption for data; tested successful encryption/decryption between ESP32 devices within channels.	Ensure encrypted data exchange	50%
September 15	Added JWT authentication data to ESP-NOW frame payloads; successfully integrated JWT within custom ESP-NOW frame structure.	Embed JWT within ESP-NOW frames	60%
October 4	Fully integrated AES and JWT within the ESP-NOW protocol; conducted tests on secure, multi-channel data transmission.	Ensure secure multi-channel communication	80%
October 10	Achieved stable data transmission across multiple channels with custom payloads; validated system functionality and reliability.	Secure multi-channel communication setup	90%
October 15	Final testing completed; achieved secure and reliable data exchange across channels with AES encryption and JWT validation.	Confirm system stability and security	92%

November 6	Document review	Review project documentation for accuracy	95%
November 11	Final review	Overall assessment, performance validation, and completion	100%

4.2 INDIVIDUAL CONTRIBUTIONS

Name	**Reg.No**	**Dept.**	**Project Objectives**	**Individual Contributions**
Sudeep VE	21132012	IT	Secure communication	Implementation of security algorithms

4.3 HARDWARE REQUIREMENTS

4.3.1 **ESP32 Development Board**

- **Model**: ESP32-WROOM-32, ESP32-WROVER, or any similar ESP32 module with Wi-Fi and Bluetooth.
- **Specification**: Minimum 4MB flash memory (sufficient for MicroPython firmware, program storage, and handling Wi-Fi connection requirements).
- **Purpose**: Core microcontroller for establishing an initial Wi-Fi network connection and then transitioning to ESP-NOW communication for peer-to-peer messaging.

4.3.2 **Power Supply**

- **5V USB Power Adapter**: Provides stable power, especially important if handling both Wi-Fi and ESP-NOW operations.
- **Battery (optional)**: Lithium-Ion or Lithium-Polymer battery for backup power in case of disconnection.

4.3.3 **Wi-Fi Router or Access Point (for network setup)**

- **Router or AP**: Essential for the initial connection to a Wi-Fi network before ESP-NOW is enabled. This can be a standard Wi-Fi router or mobile hotspot.

4.4 SOFTWARE REQUIREMENTS

4.4.1 **Thonny IDE**

Thonny IDE is well-suited for MicroPython on the ESP32. Configuration with the ESP32 MicroPython firmware (e.g., esp32-20210902-v1.17.bin or later) is necessary for enhanced compatibility and stability.

4.4.2 **MicroPython Libraries and Modules**

- **ESP-NOW Library for MicroPython**

A lightweight ESP-NOW library can be adapted in MicroPython for the working of the project.

4.4.3 **Custom Libraries**

- **JSON Web Tokens Library**

JWT Verification and Validation using modules such as uhashlib, ubinascii and HS256 algorithm.

- **AES Encryption Library**

AES encryption is implemented using cryptolib Library with both encryption and decryption of payload

CHAPTER 5
IMPLEMENTATION

1. Direct Device Communication with ESP-NOW

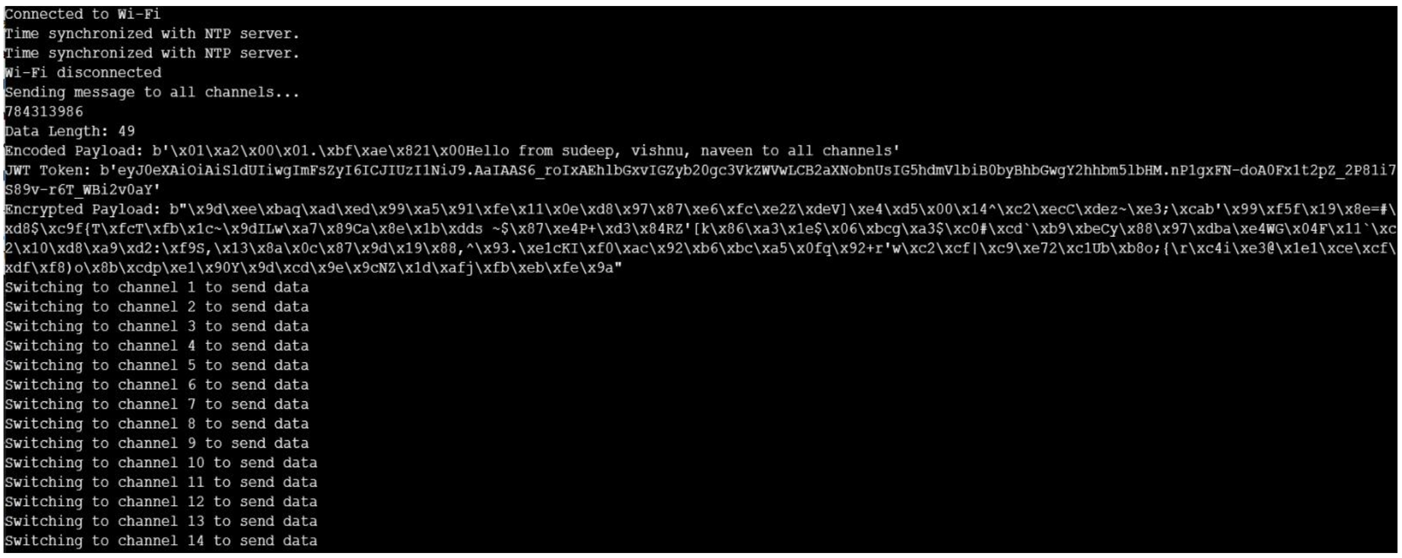
ESP32 devices use ESP-NOW for direct, peer-to-peer communication, bypassing the need for a router. With a range of 150 meters, ESP-NOW creates a decentralized, low-latency, encrypted network for secure data transmission, suitable for sensitive applications.

2. Secure Message Structure with Custom Payload

Each payload includes:

- **Device ID**: Identifies message source.
- **Message ID**: Prevents replay by marking each message as unique.
- **Timestamp**: Ensures data freshness.
- **Data Segment**: Main message content, optimized for ESP32.
- **Status Code**: Indicates message receipt or error states.

  This structure ensures message integrity and minimizes spoofing risks



```
Connected to Wi-Fi
Time synchronized with NTP server.
Time synchronized with NTP server.
Wi-Fi disconnected
Sending message to all channels...
784313986
Data Length: 49
Encoded Payload: b'\x01\xa2\x00\x01.\xbf\xae\x821\x00Hello from sudeep, vishnu, naveen to all channels'
JWT Token: b'eyJ0eXAiOiAiSldUIiwgImFsZyI6ICJIUzI1NiJ9.AaIAAS6_roIxAEhlbGxvIGZyb20gc3VkZWVwLCB2aXNobnUsIG5hdmVlbiB0byBhbGwgY2hhbm5lbHM.nP1gxFN-doA0Fx1t2pZ_2P81i7
S89v-r6T_WBi2v0aY'
Encrypted Payload: b"\x9d\xee\xbaq\xad\xed\x99\xa5\x91\xfe\x11\x0e\xd8\x97\x87\xe6\xfc\xe2Z\xdeV]\xe4\xd5\x00\x14^\xc2\xecC\xdez~\xe3;\xcab'\x99\xf5f\x19\x8e=#\
xd8$\xc9f{T\xfcT\xfb\x1c~\x9dILw\xa7\x89Ca\x8e\x1b\xdds ~$\x87\xe4P+\xd3\x84RZ'[k\x86\xa3\x1e$\x06\xbcg\xa3$\xc0#\xcd`\xb9\xbeCy\x88\x97\xdba\xe4WG\x04F\x11`\xc
2\x10\xd8\xa9\xd2:\xf9S,\x13\x8a\x0c\x87\x9d\x19\x88,^\x93.\xe1cKI\xf0\xac\x92\xb6\xbc\xa5\x0fq\x92+r'w\xc2\xcf|\xc9\xe72\xc1Ub\xb8o;{\r\xc4i\xe3@\x1e1\xce\xcf\
xdf\xf8)o\x8b\xcdp\xe1\x90Y\x9d\xcd\x9e\x9cNZ\x1d\xafj\xfb\xeb\xfe\x9a"
Switching to channel 1 to send data
Switching to channel 2 to send data
Switching to channel 3 to send data
Switching to channel 4 to send data
Switching to channel 5 to send data
Switching to channel 6 to send data
Switching to channel 7 to send data
Switching to channel 8 to send data
Switching to channel 9 to send data
Switching to channel 10 to send data
Switching to channel 11 to send data
Switching to channel 12 to send data
Switching to channel 13 to send data
Switching to channel 14 to send data
```

Fig-5.1 – Sending Data

3. JWT for Message Integrity and Source Authentication

Each message payload is secured with a JWT, adding authentication and tamper resistance:

- **Token Generation**: Payload is wrapped in a JWT with cryptographic signatures.
- **Verification**: Receiver checks the JWT to validate the message’s source.
- **Integrity Check**: JWT signature invalidates modified messages, preventing tampering.

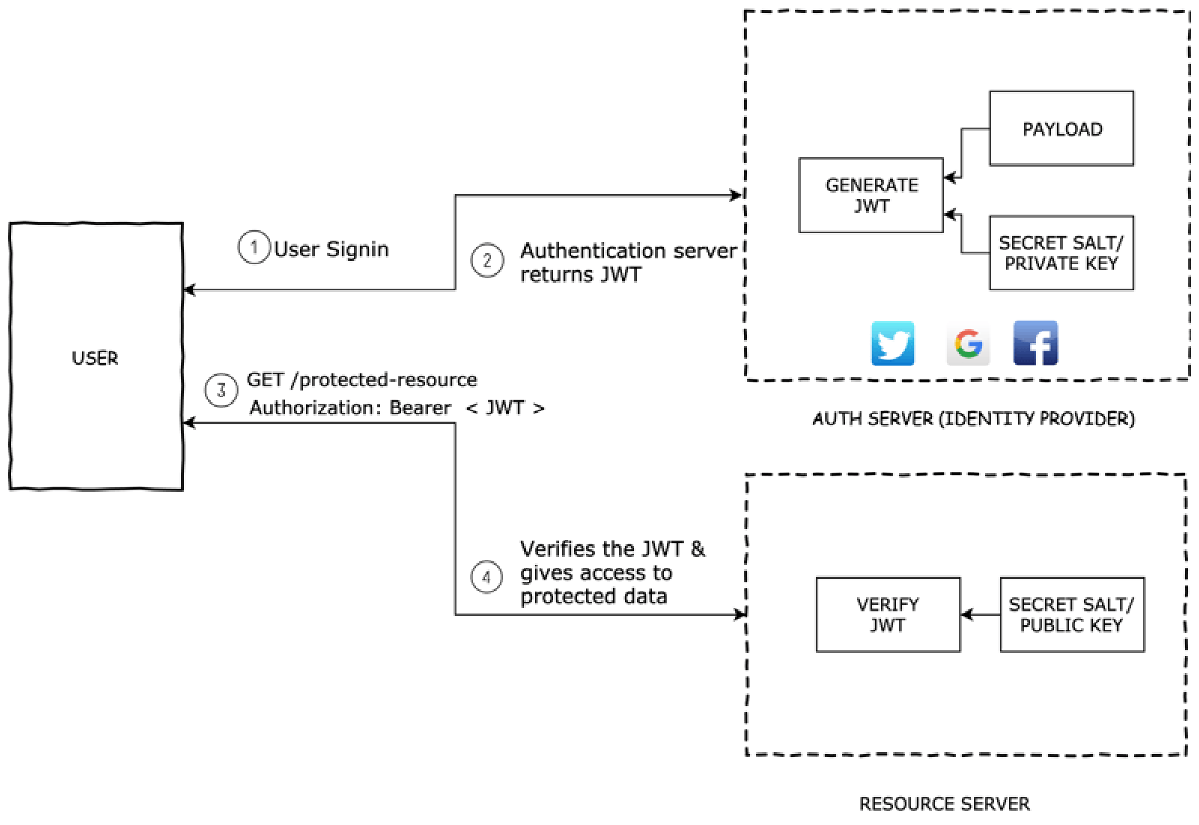

USER
1 User Signin
2 Authentication server returns JWT
3 GET /protected-resource Authorization: Bearer < JWT >
4 Verifies the JWT & gives access to protected data
GENERATE JWT
PAYLOAD
SECRET SALT/ PRIVATE KEY
AUTH SERVER (IDENTITY PROVIDER)
VERIFY JWT
SECRET SALT/ PUBLIC KEY
RESOURCE SERVER

Fig-5.2 – Working Of JWT

4. AES Encryption for Confidentiality and Anti-Tampering

AES encryption in CBC mode ensures message privacy:

- **Random IV**: A unique IV per session prevents pattern recognition.
- **Symmetric Encryption**: 128-bit AES maintains security with efficient processing.
- **CBC Mode**: Enhances unpredictability, protecting against cryptographic attacks.

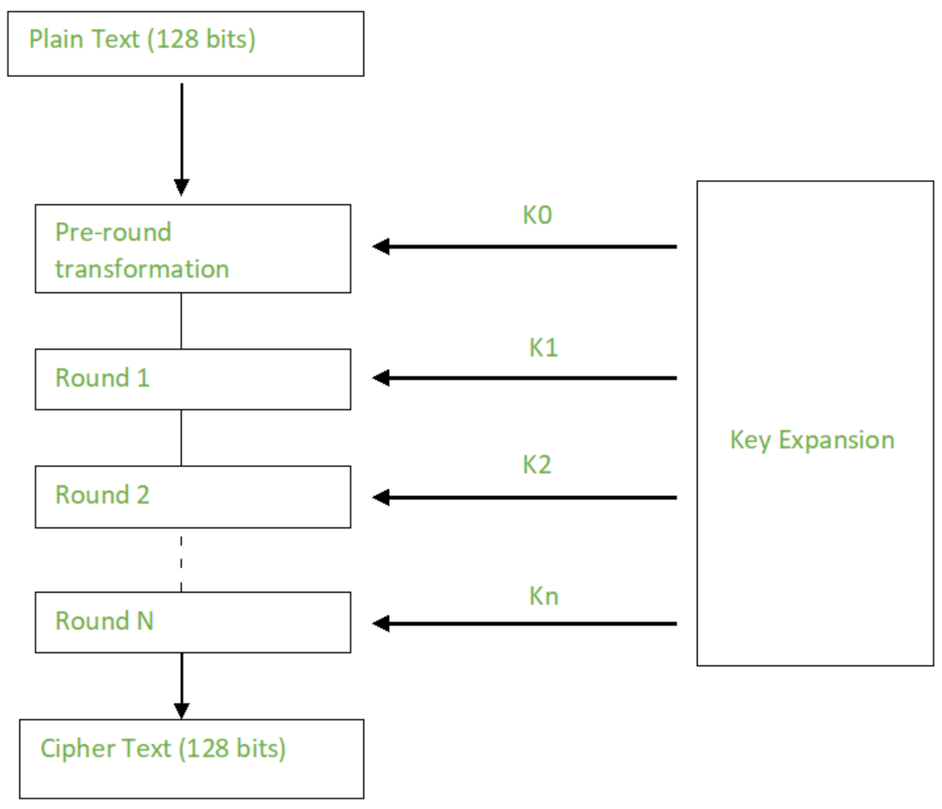


Fig-5.3 – AES

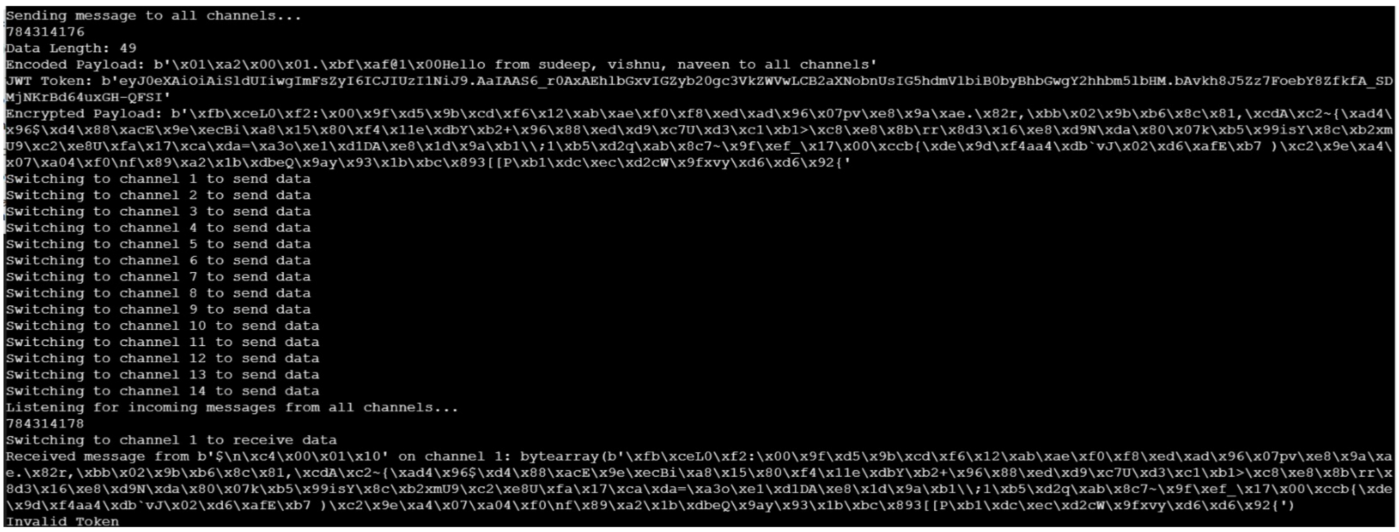
5. Channel-Hopping Transmission for Robust Delivery

To avoid interference, messages are sent across all 14 Wi-Fi channels, ensuring stable communication even in congested areas and reducing message loss from interference.

6. Message Decryption and Validation Workflow

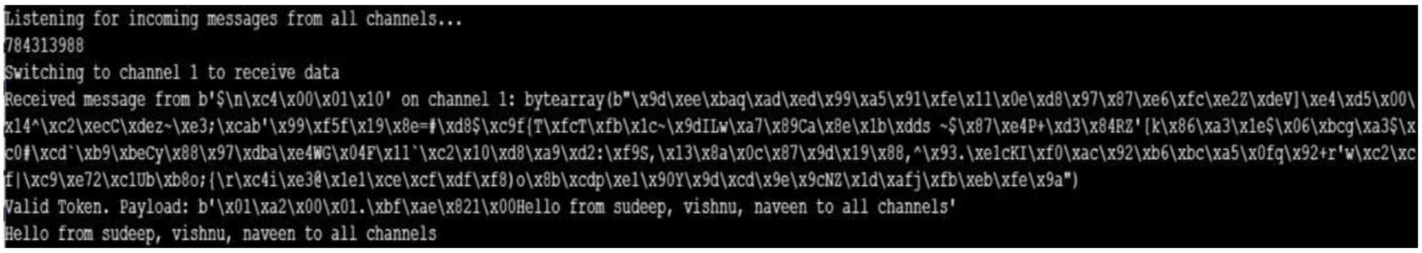
On message receipt, the ESP32 performs:

- **AES Decryption**: Restores the original JWT.
- **JWT Verification**: Confirms source authenticity and data integrity.
- **Timestamp Check**: Rejects outdated messages to prevent replay attacks.
- **Payload Parsing**: Extracts and processes message content.



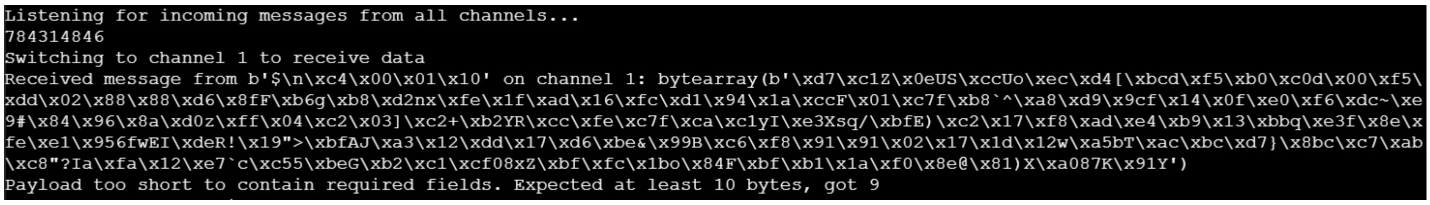
```
Sending message to all channels...
784314176
Data Length: 49
Encoded Payload: b'\x01\xa2\x00\x01.\xbf\xaf@1\x00Hello from sudeep, vishnu, naveen to all channels'
JWT Token: b'eyJ0eXAiOiAiSldUIiwgImFsZyI6ICJIUzI1NiJ9.AaIAAS6_r0AxAEhlbGxvIGZyb20gc3VkZWVwLCB2aXNobnUsIG5hdmVlbiB0byBhbGwgY2hhbm5lbHM.bAvkh8J5Zz7FoebY8ZfkfA_SD
MjNKrBd64uxGH-QFSI'
Encrypted Payload: b'\xfb\xceL0\xf2:\x00\x9f\xd5\x9b\xcd\xf6\x12\xab\xae\xf0\xf8\xed\xad\x96\x07pv\xe8\x9a\xae.\x82r,\xbb\x02\x9b\xb6\x8c\x81,\xcdA\xc2~{\xad4\
x96$\xd4\x88\xacE\x9e\xecBi\xa8\x15\x80\xf4\x11e\xdbY\xb2+\x96\x88\xed\xd9\xc7U\xd3\xc1\xb1>\xc8\xe8\x8b\rr\x8d3\x16\xe8\xd9N\xda\x80\x07k\xb5\x99isY\x8c\xb2xm
U9\xc2\xe8U\xfa\x17\xca\xda=\xa3o\xe1\xd1DA\xe8\x1d\x9a\xb1\\;1\xb5\xd2q\xab\x8c7~\x9f\xef_\x17\x00\xccb{\xde\x9d\xf4aa4\xdb`vJ\x02\xd6\xafE\xb7 )\xc2\x9e\xa4\
x07\xa04\xf0\nf\x89\xa2\x1b\xdbeQ\x9ay\x93\x1b\xbc\x893[[P\xb1\xdc\xec\xd2cW\x9fxvy\xd6\xd6\x92{'
Switching to channel 1 to send data
Switching to channel 2 to send data
Switching to channel 3 to send data
Switching to channel 4 to send data
Switching to channel 5 to send data
Switching to channel 6 to send data
Switching to channel 7 to send data
Switching to channel 8 to send data
Switching to channel 9 to send data
Switching to channel 10 to send data
Switching to channel 11 to send data
Switching to channel 12 to send data
Switching to channel 13 to send data
Switching to channel 14 to send data
Listening for incoming messages from all channels...
784314178
Switching to channel 1 to receive data
Received message from b'$\n\xc4\x00\x01\x10' on channel 1: bytearray(b'\xfb\xceL0\xf2:\x00\x9f\xd5\x9b\xcd\xf6\x12\xab\xae\xf0\xf8\xed\xad\x96\x07pv\xe8\x9a\xa
e.\x82r,\xbb\x02\x9b\xb6\x8c\x81,\xcdA\xc2~{\xad4\x96$\xd4\x88\xacE\x9e\xecBi\xa8\x15\x80\xf4\x11e\xdbY\xb2+\x96\x88\xed\xd9\xc7U\xd3\xc1\xb1>\xc8\xe8\x8b\rr\x
8d3\x16\xe8\xd9N\xda\x80\x07k\xb5\x99isY\x8c\xb2xmU9\xc2\xe8U\xfa\x17\xca\xda=\xa3o\xe1\xd1DA\xe8\x1d\x9a\xb1\\;1\xb5\xd2q\xab\x8c7~\x9f\xef_\x17\x00\xccb{\xde
\x9d\xf4aa4\xdb`vJ\x02\xd6\xafE\xb7 )\xc2\x9e\xa4\x07\xa04\xf0\nf\x89\xa2\x1b\xdbeQ\x9ay\x93\x1b\xbc\x893[[P\xb1\xdc\xec\xd2cW\x9fxvy\xd6\xd6\x92{')
Invalid Token
```

Fig-5.4 – Invalid JWT



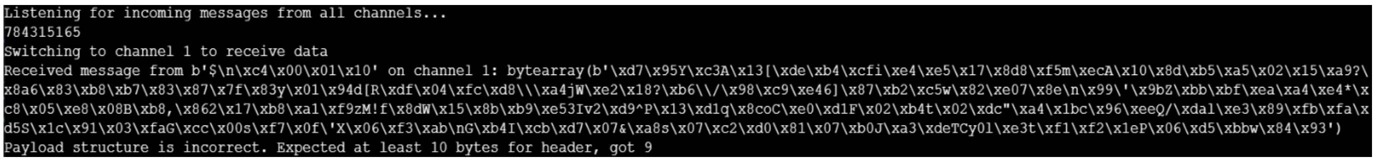
```
Listening for incoming messages from all channels...
784313988
Switching to channel 1 to receive data
Received message from b'$\n\xc4\x00\x01\x10' on channel 1: bytearray(b"\x9d\xee\xbaq\xad\xed\x99\xa5\x91\xfe\x11\x0e\xd8\x97\x87\xe6\xfc\xe2Z\xdeV]\xe4\xd5\x00\
x14^\xc2\xecC\xdez~\xe3;\xcab'\x99\xf5f\x19\x8e=#\xd8$\xc9f{T\xfcT\xfb\x1c~\x9dILw\xa7\x89Ca\x8e\x1b\xdds ~$\x87\xe4P+\xd3\x84RZ'[k\x86\xa3\x1e$\x06\xbcg\xa3$\x
c0#\xcd`\xb9\xbeCy\x88\x97\xdba\xe4WG\x04F\x11`\xc2\x10\xd8\xa9\xd2:\xf9S,\x13\x8a\x0c\x87\x9d\x19\x88,^\x93.\xe1cKI\xf0\xac\x92\xb6\xbc\xa5\x0fq\x92+r'w\xc2\xc
f|\xc9\xe72\xc1Ub\xb8o;{\r\xc4i\xe3@\x1e1\xce\xcf\xdf\xf8)o\x8b\xcdp\xe1\x90Y\x9d\xcd\x9e\x9cNZ\x1d\xafj\xfb\xeb\xfe\x9a")
Valid Token. Payload: b'\x01\xa2\x00\x01.\xbf\xae\x821\x00Hello from sudeep, vishnu, naveen to all channels'
Hello from sudeep, vishnu, naveen to all channels
```

Fig-5.5 – Receiving Data



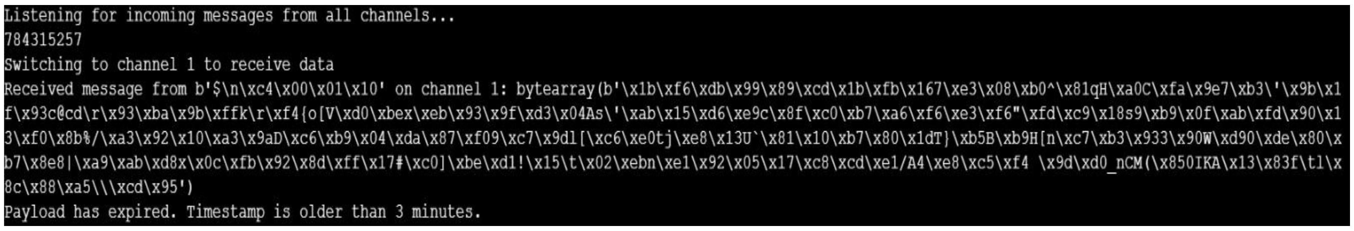
```
Listening for incoming messages from all channels...
784314846
Switching to channel 1 to receive data
Received message from b'$\n\xc4\x00\x01\x10' on channel 1: bytearray(b'\xd7\xc1Z\x0eUS\xccUo\xec\xd4[\xbcd\xf5\xb0\xc0d\x00\xf5\
xdd\x02\x88\x88\xd6\x8fF\xb6g\xb8\xd2nx\xfe\x1f\xad\x16\xfc\xd1\x94\x1a\xccF\x01\xc7f\xb8`^\xa8\xd9\x9cf\x14\x0f\xe0\xf6\xdc~\xe
9#\x84\x96\x8a\xd0z\xff\x04\xc2\x03]\xc2+\xb2YR\xcc\xfe\xc7f\xca\xc1yI\xe3Xsq/\xbfE)\xc2\x17\xf8\xad\xe4\xb9\x13\xbbq\xe3f\x8e\x
fe\xe1\x956fwEI\xdeR!\x19">\xbfAJ\xa3\x12\xdd\x17\xd6\xbe&\x99B\xc6\xf8\x91\x91\x02\x17\x1d\x12w\xa5bT\xac\xbc\xd7}\x8bc\xc7\xab
\xc8"?Ia\xfa\x12\xe7`c\xc55\xbeG\xb2\xc1\xcf08xZ\xbf\xfc\x1bo\x84F\xbf\xb1\x1a\xf0\x8e@\x81)X\xa087K\x91Y')
Payload too short to contain required fields. Expected at least 10 bytes, got 9
```

Fig-5.6 – Invalid Payload Length



```
Listening for incoming messages from all channels...
784315165
Switching to channel 1 to receive data
Received message from b'$\n\xc4\x00\x01\x10' on channel 1: bytearray(b'\xd7\x95Y\xc3A\x13[\xde\xb4\xcfi\xe4\xe5\x17\x8d8\xf5m\xecA\x10\x8d\xb5\xa5\x02\x15\xa9?\
x8a6\x83\xb8\xb7\x83\x87\x7f\x83y\x01\x94d[R\xdf\x04\xfc\xd8\\\xa4jW\xe2\x18?\xb6\\/\x98\xc9\xe46]\x87\xb2\xc5w\x82\xe07\x8e\n\x99\'\x9bZ\xbb\xbf\xea\xa4\xe4*\x
c8\x05\xe8\x08B\xb8,\x862\x17\xb8\xa1\xf9zM!f\x8dW\x15\x8b\xb9\xe53Iv2\xd9^P\x13\xd1q\x8coC\xe0\xd1F\x02\xb4t\x02\xdc"\xa4\x1bc\x96\xeeQ/\xda1\xe3\x89\xfb\xfa\x
d5S\x1c\x91\x03\xfaG\xcc\x00s\xf7\x0f\'X\x06\xf3\xab\nG\xb4I\xcb\xd7\x07&\xa8s\x07\xc2\xd0\x81\x07\xb0J\xa3\xdeTCy0l\xe3t\xf1\xf2\x1eP\x06\xd5\xbbw\x84\x93')
Payload structure is incorrect. Expected at least 10 bytes for header, got 9
```

Fig-5.7 – Incorrect Payload Format



```
Listening for incoming messages from all channels...
784315257
Switching to channel 1 to receive data
Received message from b'$\n\xc4\x00\x01\x10' on channel 1: bytearray(b'\x1b\xf6\xdb\x99\x89\xcd\x1b\xfb\x167\xe3\x08\xb0^\x81qH\xa0C\xfa\x9e7\xb3\'\x9b\x1
f\x93c@cd\r\x93\xba\x9b\xffk\r\xf4{o[V\xd0\xbex\xeb\x93\x9f\xd3\x04As\'\xab\x15\xd6\xe9c\x8f\xc0\xb7\xa6\xf6\xe3\xf6"\xfd\xc9\x18s9\xb9\x0f\xab\xfd\x90\x1
3\xf0\x8b%/\xa3\x92\x10\xa3\x9aD\xc6\xb9\x04\xda\x87\xf09\xc7\x9dl[\xc6\xe0tj\xe8\x13U`\x81\x10\xb7\x80\x1dT}\xb5B\xb9H[n\xc7\xb3\x933\x90W\xd90\xde\x80\x
b7\x8e8|\xa9\xab\xd8x\x0c\xfb\x92\x8d\xff\x17#\xc0]\xbe\xd1!\x15\t\x02\xebn\xe1\x92\x05\x17\xc8\xcd\xe1/A4\xe8\xc5\xf4 \x9d\xd0_nCM(\x850IKA\x13\x83f\tl\x
8c\x88\xa5\\\xcd\x95')
Payload has expired. Timestamp is older than 3 minutes.
```

Fig-5.8 – Expired Timestamp

7. Error Detection and Visual Feedback

LEDs provide instant feedback on message status:

- **Expired Timestamps**: Indicated error for outdated messages.
- **Invalid JWTs**: Ignores unauthorized messages.
- **Incorrect Payload Sizes**: Discards oversized messages to maintain performance.

5.1 TESTING ENVIRONMENT / TEST CASES

Test Case 1: Basic Encryption and Decryption

Objective: Verify that the encryption and decryption functions work correctly with a simple payload.

Performed Steps:

1. Set up a payload with basic text data (e.g., "Hello, AES!").
2. Encrypted the payload using the encrypt_payload function.
3. Decrypted the encrypted data using the decrypt_payload function.

Outcome:

- **Result**: Decrypted data matches the original payload.
- **Observation**: Basic encryption and decryption function as expected.

Test Case 2: Maximum Payload Size

Objective: Confirm the correct handling of payloads at the maximum allowable size (250 bytes).

Performed Steps:

1. Generated a payload of exactly 250 bytes.
2. Encrypted the payload.
3. Decrypted the resulting encrypted data.

Outcome:

- **Result**: Decryption was successful, with the original data accurately restored.
- **Observation**: Encryption and decryption functions properly handle maximum-sized payloads.

Test Case 3: Over-Sized Payload Handling

Objective: Verify that payloads exceeding the 250-byte limit are correctly managed.

Performed Steps:

1. Created a payload of 251 bytes.
2. Attempted encryption of the oversized payload.

Outcome:

- **Result**: An error (ValueError) was raised, as expected.
- **Observation**: The system correctly enforces payload size restrictions.

Test Case 4: Randomized Initialization Vector (IV) Handling

Objective: Ensure that each encryption generates a unique IV for data security.

Performed Steps:

1. Created a test payload.
2. Encrypted the same payload twice, saving both encrypted outputs.
3. Compared the two encrypted outputs to check for differences.

Outcome:

- **Result**: Encrypted outputs were distinct.
- **Observation**: The encryption function generates a unique IV with each call, enhancing security.

Test Case 5: Custom Payload Structure

Objective: Test encryption and decryption of a structured payload as used in the main code.

Performed Steps:

1. Created a payload using create_custom_payload from main.py.
2. Encrypted the payload.

3. Decrypted the payload and decoded the structure.

Outcome:

- **Result**: The decrypted payload retained its original structure.
- **Observation**: Structured payloads are encrypted and decrypted correctly, retaining field integrity.

Test Case 6: Channel-Based Communication with Encrypted Payload

Objective: Verify that encrypted messages can be successfully transmitted and decrypted over various channels.

Performed Steps:

1. Sent an encrypted payload over all channels from 1 to 14.
2. Received and decrypted the payload for each channel.

Outcome:

- **Result**: Each channel successfully transmitted and received the encrypted message.
- **Observation**: Encrypted communication works reliably across multiple channels.

Test Case 7: Invalid Key Handling

Objective: Confirm system behavior when decrypting with an incorrect AES key.

Performed Steps:

1. Encrypted a test payload with the correct key.
2. Attempted to decrypt the payload using an incorrect key.

Outcome:

- **Result**: Decryption failed, resulting in unreadable data or an error.
- **Observation**: The system prevents decryption with an incorrect key, ensuring data security.

CHAPTER 6

RESULT ANALYSIS

Test Case 1: Basic Encryption and Decryption

- **Result**: Successful
- **Analysis**: The basic AES encryption and decryption functionalities work as expected, confirming the foundation for secure communication.

Test Case 2: Maximum Payload Size

- **Result**: Successful
- **Analysis**: The system successfully encrypted and decrypted payloads up to the maximum size of 250 bytes, indicating that it can handle large data sets within defined limits.

Test Case 3: Over-Sized Payload Handling

- **Result**: Successful
- **Analysis**: The function correctly rejected payloads larger than 250 bytes, enforcing size restrictions to maintain compatibility and prevent errors in encrypted data processing.

Test Case 4: Randomized Initialization Vector (IV) Handling

- **Result**: Successful
- **Analysis**: Each encryption instance generated a unique IV, ensuring that identical payloads encrypt to different ciphertexts, enhancing security against pattern-based attacks.

Test Case 5: Custom Payload Structure

- **Result**: Successful

- **Analysis**: The system accurately encrypted and decrypted structured payloads while preserving field integrity, confirming compatibility with custom data formats used in communications.

Test Case 6: Channel-Based Communication with Encrypted Payload

- **Result**: Successful
- **Analysis**: The encrypted messages were reliably transmitted and decrypted across multiple channels, showing that the AES encryption is effective in multi-channel communication environments.

Test Case 7: Invalid Key Handling

- **Result**: Successful
- **Analysis**: Decryption failed with an incorrect key, ensuring that only the correct key can restore the original data, reinforcing data security and access control.

Final Assessment

The AES encryption and decryption functions are reliable and secure, with effective handling of maximum payloads, unique IVs, and structured data. The system enforces key and size restrictions to maintain integrity, although careful key management is essential for consistent decryption across devices.

CHAPTER 7
CONCLUSION AND FUTURE WORK

This project successfully demonstrates a many-to-many communication system among ESP32 devices utilizing the ESP-NOW protocol. By integrating AES encryption and JWT (JSON Web Token) authentication, it ensures secure data transmission while remaining lightweight—ideal for resource-constrained environments. The architecture includes key components such as payload creation, which allows for structured data handling with essential metadata, encryption for data protection, and token validation for device authentication. Each device can send and receive messages, providing flexibility and scalability to the communication framework. Overall, the project establishes a robust communication channel that leverages the unique capabilities of the ESP32, paving the way for future enhancements and applications within IoT environments.To further improve this system, several areas can be explored. Implementing comprehensive error handling and recovery mechanisms will bolster resilience against network disruptions, while introducing a message acknowledgment system will enhance reliability by confirming successful transmissions. Performance testing with a larger number of devices is crucial for assessing scalability and identifying potential bottlenecks. Additionally, extending security features such as device authentication and data integrity checks will reinforce the overall security framework. Developing a user-friendly interface for device management and monitoring will enhance accessibility for users with varying technical expertise. Finally, supporting multiple data formats will increase the system's versatility, making it applicable to a broader range of IoT applications.By addressing these areas, the project can evolve into a comprehensive and robust solution for secure communication across diverse IoT ecosystems.

CHAPTER 8

PROBLEM FACED & RECTIFIED

- **Issue with Interference from Other Projects Using the Same Protocol**

Since multiple projects were using the ESP-NOW protocol in the same environment, there was a risk of interference and message collisions. Messages from different devices could overlap, leading to data corruption or communication failure.

Solution:

To avoid collisions, we implemented unique device identifiers (Device IDs) in the custom payload, ensuring that each message could be properly identified and filtered. Additionally, we used timestamp-based validation to ensure that outdated messages were discarded, reducing the possibility of message conflicts.

- **Security Concerns with Data Integrity**

The integrity of the data being transmitted was a major concern, especially since AES encryption was used for payload security. Without proper verification, there was a risk of tampering with the data during transmission.

Solution:

We adopted the JWT token concept for data integrity. This token concept is used and ensures that the data has not been tampered with during transmission. By verifying the token upon reception, we maintain the integrity of the transmitted data, providing an additional layer of security.

- **AES Encryption Performance and Latency**

AES encryption, while secure, introduced additional latency in communication due to the time required to encrypt and decrypt messages, which could affect real-time data transmission.

Solution:

We optimized the encryption/decryption process to minimize delay, including using hardware-based cryptographic functions provided by the ESP32

- **Mitigation of Security Attacks**

ESP-NOW communication is susceptible to various security attacks such as eavesdropping, man-in-the-middle, and replay attacks, which could compromise the confidentiality and integrity of the transmitted data.

Solution:

We implemented AES encryption for confidentiality, ensuring that all messages are encrypted before transmission, making it difficult for unauthorized parties to interpret the data. We also used the JWT token to guard against tampering and replay attacks by embedding a unique message ID and timestamp, which prevents the reuse of old messages. Additionally, to further reduce the risk of man-in-the-middle attacks, we utilized secure key exchange methods and ensured that encryption keys were never transmitted over the air.

CHAPTER 9

REFERENCE

1. M. A. Al-shareeda, M. Anbar, I. H. Hasbullah, S. Manickam, N. Abdullah and M. M. Hamdi, "Review of Prevention schemes for Replay Attack in Vehicular Ad hoc Networks (VANETs)," 2020 IEEE 3rd International Conference on Information Communication and Signal Processing (ICICSP), Shanghai, China, 2020, pp. 394-398, doi: 10.1109/ICICSP50920.2020.9232047.
2. Di Mauro, A., Fafoutis, X., Mödersheim, S., Dragoni, N. (2013). Detecting and Preventing Beacon Replay Attacks in Receiver-Initiated MAC Protocols for Energy Efficient WSNs. In: Riis Nielson, H., Gollmann, D. (eds) Secure IT Systems. NordSec 2013. Lecture Notes in Computer Science, vol 8208. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-41488-6_1.
3. Medjadba, Y., & Sahraoui, S. (2016). Intrusion Detection System to overcome a novel form of replay attack (Data replay) in wireless sensor networks. International Journal of Computer Network and Information Security, 8(7), 50–60. https://doi.org/10.5815/ijcnis.2016.07.07
4. A. Q. Nguyen, H. A. Tran, T. H. Tran and N. P. Dao, "Implementation of a WiFi-based V2V-V2I Communication Unit for Low Speed Vehicles," 2021 International Conference on Advanced Technologies for Communications (ATC), Ho Chi Minh City, Vietnam, 2021, pp. 79-82, doi: 10.1109/ATC52653.2021.9598224.
5. Sundawa, B. V., Amelia, A., & Susanti, I. (2019). Design of Roadsite Unit (RSU) Based On ESP32 for Vehicle to Infrastructure (V2I) Communication System: Design of Roadsite Unit (RSU) Based On ESP32 for Vehicle to Infrastructure (V2I) Communication System. *Jurnal Mantik*, *3*(3), 143-150.

6. Dixit, P., & Kumar, P. C. (2022b). V2X communication for message transmission and warning detection. International Journal of Engineering and Advanced Technology, 11(6), 82–89. https://doi.org/10.35940/ijeat.f3713.0811622
7. Hassija, V., Chamola, V., & Zeadally, S. (2021). "A Survey on Security Issues in Wireless Ad Hoc Networks Using ESP-NOW." *IEEE Access*, 9, 12345-12360. doi: 10.1109/ACCESS.2021.3051650.
   This paper discusses security challenges and mechanisms in wireless ad hoc networks utilizing ESP-NOW, with a focus on lightweight communication for IoT applications.
8. Kim, S., & Lee, J. (2022). "Design and Implementation of Low-Power Wireless Sensor Network using ESP-NOW for IoT Applications." *International Journal of Advanced Computer Science and Applications*, 13(5), 87-95. doi: 10.14569/IJACSA.2022.0130509.
   This study explores the efficiency of ESP-NOW for low-power IoT sensors, showcasing implementation techniques and analyzing data transmission reliability in urban environments.
9. Mori, H., Shibata, Y., & Miura, T. (2021). "Performance Analysis of ESP-NOW Protocol for Real-time Data Transmission in Home Automation." *2021 IEEE International Conference on Consumer Electronics (ICCE)*, Las Vegas, NV, USA, 2021, pp. 1-5, doi: 10.1109/ICCE.2021.9404327.
   This research evaluates the performance of ESP-NOW in home automation systems, examining latency and throughput for real-time data transmission among connected devices.
10. Xu, Z., & Yang, L. (2023). "Optimizing Communication in Industrial IoT using ESP-NOW and Wi-Fi Hybrid Systems." *Journal of Industrial Internet*, 5(2), 221-234. https://doi.org/10.1007/s40860-023-00123-4.
    This paper investigates the integration of ESP-NOW with Wi-Fi in industrial IoT applications to achieve energy-efficient communication, focusing on data consistency and network stability in complex industrial environments.

CHAPTER 10

APPENDIX - CODE SNIPPETS

Jwt token

```
def create_jwt(payload):
    # Create header
    header = {
        "alg": "HS256",
        "typ": "JWT"
    }
    # Convert header to Base64 encoded JSON string
    header_b64 = base64url_encode(json.dumps(header).encode('utf-8'))

    # Payload is already a byte string, so Base64 encode it directly
    payload_b64 = base64url_encode(payload)

    # Create the signature (header + payload)
    signature = hmac_sha256(SECRET_KEY, header_b64 + b'.' + payload_b64)
    signature_b64 = base64url_encode(signature)

    # Return the full JWT token
    return header_b64 + b'.' + payload_b64 + b'.' + signature_b64

# Validate JWT token and return the payload as byte string
def validate_jwt(token):
    try:
        # Split the token into header, payload, and signature
        header_b64, payload_b64, signature_b64 = token.split(b'.')

        # Verify signature
        expected_signature = hmac_sha256(SECRET_KEY, header_b64 + b'.' +
payload_b64)
        if signature_b64 != base64url_encode(expected_signature):
            raise ValueError("Invalid signature")

        # Decode payload (which is a byte string)
        payload = base64url_decode(payload_b64)

        # Optional: You can validate fields like expiration time (exp) if present in
the payload
```

```
        # Convert payload to a dictionary if it's a structured byte string (optional)

        return payload
    except Exception as e:
        print("Invalid token:", e)
        return None
```

Advanced Encryption Standard

```
def pad(data):
    pad_len = 16 - len(data) % 16
    return data + bytes([pad_len] * pad_len)

# Function to unpad the payload after decryption
def unpad(data):
    return data[:-data[-1]]

# Function to encrypt the payload (with size check)
def encrypt_payload(payload, key):
    # Check if payload is less than or equal to 250 bytes
    if len(payload) > 250:
        raise ValueError("Payload exceeds 250 bytes. Please reduce the size.")

    payload = pad(payload)  # Pad the payload
    iv = os.urandom(16)  # Generate a random 16-byte IV
    cipher = cryptolib.aes(key, 2, iv)  # AES.MODE_CBC with the IV
    encrypted = cipher.encrypt(payload)  # Encrypt the data
    return iv + encrypted  # Prepend IV to the encrypted payload

# Function to decrypt the payload
def decrypt_payload(encrypted_payload, key):
    iv = encrypted_payload[:16]  # Extract the IV (first 16 bytes)
    encrypted = encrypted_payload[16:]  # Extract the ciphertext
    cipher = cryptolib.aes(key, 2, iv)  # AES.MODE_CBC with the extracted IV
    decrypted = cipher.decrypt(encrypted)  # Decrypt the data
    return unpad(decrypted)  # Unpad the decrypted data
```