RASPBERRY HOME – SECURE PRIVATE NETWORK FOR IoT DEVICES

A Dissertation submitted in partial fulfillment of the requirements for the award of the Degree of

BACHELOR OF ENGINEERING
IN
ELECTRONICS AND COMMUNICATION ENGINEERING

BY

MOHAMMED WAJAHATH HASEEB	**1604-21-735-091**
SHAIK ABDUL SAMI	**1604-21-735-093**
MOHAMMED ABDUL RAHMAN	**1604-21-735-096**

Department of Electronics and Communication Engineering

Muffakham Jah College of Engineering and Technology

Banjara Hills, Hyderabad – 500 034

(Affiliated to Osmania University)

JUNE 2025

RASPBERRY HOME – SECURE PRIVATE NETWORK FOR IoT DEVICES

A Dissertation submitted in partial fulfillment of the requirements for the award of the Degree of

BACHELOR OF ENGINEERING
IN
ELECTRONICS AND COMMUNICATION ENGINEERING

BY

MOHAMMED WAJAHATH HASEEB	**1604-21-735-091**
SHAIK ABDUL SAMI	**1604-21-735-093**
MOHAMMED ABDUL RAHMAN	**1604-21-735-096**

Under the guidance of
Dr. SALMA FAUZIA
Assistant Professor, Department of ECE, MJCET

Department of Electronics and Communication Engineering

Muffakham Jah College of Engineering and Technology

Banjara Hills, Hyderabad – 500 034

(Affiliated to Osmania University)

JUNE 2025

MUFFAKHAM JAH
COLLEGE OF ENGINEERING &TECHNOLOGY
(Est. by Sultan-Ul-Uloom Education Society in 1980)
(Affiliated to Osmania University, Hyderabad)
Approved by the AICTE & Accredited by NBA

CERTIFICATE

This is to certify that the dissertation titled **'Raspberry Home – Secure Private Network for IoT Devices'** submitted by **Mohammed Wajahath Haseeb (Roll No: 1604-21-735-091), Shaik Abdul Sami (Roll No: 1604-21-735-093), Mohammed Abdul Rahman (Roll No: 1604-20-735-096)** in partial fulfillment of the requirements for the award of the Degree of **Bachelor of Engineering,** is a Bonafide record of work carried out by them under my guidance and supervision during the academic year 2024-2025.

The results embodied in this thesis have not been submitted to any University or Institute for the award of any Degree or Diploma.

Dr. Salma Fauzia
Project Supervisor,
Assistant Professor, ECED,

Dr. Ayesha Naaz
Professor & Head of Department, ECED,
MJCET, Hyderabad MJCET, Hyderabad

INTERNAL EXAMINER **EXTERNAL EXAMINER**

8-2-49, Mount Pleasant, Road No.3, Banjara Hills, Hyderabad - 500 034 Phone: 040-23350523, 23352084, Fax: 040-2335 3248
Website: www.mjcollege.ac.in, email: principal@mjcollege.ac.in

DECLARATION

We hereby declare that the work presented in this dissertation entitled **Raspberry Home – Secure Private Network for IoT Devices'** submitted in partial fulfillment of the requirement for the award of the Degree of **Bachelor of Engineering**, in the Department of Electronics and Communication Engineering, Muffakham Jah College of Engineering and Technology, Hyderabad is an authentic record of our own work carried out from October 2024 to May 2025 under the guidance and supervision of Ms.Dr Salma Fauzia, Assistant Professor, Department of ECE, MJCET.

We have not submitted the matter embodied in this dissertation for the award of any other degree or diploma.

Mohammed Wajahath Haseeb

Shaik Abdul Sami

Mohammed Abdul Rahman

ACKNOWLEDGEMENT

At this moment of accomplishment, we are presenting our work with great pride and pleasure. We would like to express our sincere gratitude to all those who helped us in the successful completion of our venture.

We are very much indebted and express our deep sense of gratitude to **Dr. Salma Fauzia (Assistant Professor ECED, MJCET),** our project supervisor, who took keen interest throughout the course of the project. Our zeal was kept alive throughout the project, and she provided much-needed guidance and unflinching support to complete the project. Without her constant encouragement and guidance this dissertation work could not have taken the present shape.

We sincerely thank the Project Coordinators: Dr. Kaleem Fatima, Professor, ECE Department, MJCET, and Mr. Iftekharuddin, Associate Professor, Department of ECE, for the in-depth and tough appraisals conducted by them and for their timely and qualitative suggestions that truly helped in improving the quality of the project work. More importantly the thesis writing guidelines provided by them was a blessing in disguise to write the thesis in proper structure and format overcoming major drawbacks.

We would like to thank **Dr. Ayesha Naaz**, Professor and Head, Department of Electronics and Communication Engineering, for his endless support and guidance throughout the course. All his valuable suggestions and feedback have helped us a lot to mold ourselves. He has also been a great source of inspiration in terms of work and studies.

We also extend our deepest sense of gratitude and thank **Dr. Mahipal Singh Rawat,**
Principal MJCET, for his constant support and encouragement

We also thank other faculty members of the Department of Electronics and Communication Engineering for their valuable suggestions in the project reviews and for their invaluable support and guidance throughout the engineering career. We also thank the entire non-teaching staff of the Department of Electronics and Communication Engineering for their help and support during the project and throughout the engineering career.

We would also like to thank our parents and friends for their overwhelming and wholehearted encouragement and support without which this would not have been successful.

Above all we thank the Almighty for constantly motivating us with His blessings and giving us courage at each stride to step forward with confidence and self –belief.

Mohammed Wajahat Haseeb
Shaik Abdul Sami
Mohammed Abdul Rahman

Table of Contents

ABSTRACT

The development of Raspberry Home – Secure Private Network For Iot Devices, an IoT architecture focused on enhancing security, privacy, and manageability for small-scale deployments. The system employs a Raspberry Pi as a Layer 3 gateway to create a fully isolated private network for IoT devices such as ESP8266 and ESP32, ensuring that these devices operate without direct internet access, thereby minimizing security risks.

The architecture combines several key components: network isolation using dnsmasq for DHCP and IP assignment, firewall rules configured through iptables and Uncomplicated Firewall (UFW), and a secure Over-the-Air (OTA) firmware update mechanism hosted on an HTTPS-enabled Apache server running locally on the Pi. Firmware updates are uploaded remotely using WinSCP and applied automatically via version control on connected devices.

Encrypted communication is established through a local Mosquitto MQTT broker using TLS to protect data confidentiality and integrity. Although full client-server certificate authentication was limited by the hardware capabilities of ESP8266 devices, future improvements are anticipated. Remote access to the private network is facilitated by Tailscale VPN, allowing encrypted, zero-trust connectivity without exposing devices to public networks or requiring port forwarding.

This report details the methodology, implementation, and testing of the system, demonstrating its effectiveness as a secure, cloud-independent IoT solution suited for home labs, educational environments, and secure IoT deployments.

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Secure and reliable connectivity has become a cornerstone in the rapidly expanding world of the Internet of Things (IoT), where countless small devices communicate to automate, monitor, and enhance our daily lives. These connected devices, ranging from sensors and smart appliances to industrial controllers, require a network infrastructure that not only supports efficient communication but also protects against growing cybersecurity threats. The need for secure, autonomous, and manageable IoT environments has driven innovation beyond traditional cloud-dependent architectures toward localized edge-based solutions [1].

The evolution of IoT networking has been shaped by increasing demands for privacy, security, and control. Early IoT deployments relied heavily on public internet connectivity and cloud platforms, often exposing devices to vulnerabilities such as unauthorized access, data breaches, and service disruptions. Advances in networking, cryptography, and embedded systems have paved the way for more robust architectures that emphasize network isolation, encrypted communication, and secure remote management.

Furthermore, integrating IoT systems with secure edge gateways, like the Raspberry Pi, enables the creation of private, isolated networks that prevent direct internet exposure. This approach not only mitigates risks but also enhances performance by processing data locally. Combining these gateways with secure firmware update mechanisms and VPN-based remote access establishes a comprehensive framework for resilient IoT deployments. As IoT continues to permeate various domains, such secure, autonomous networks are critical for enabling trustworthy and scalable device ecosystems.

1.1 OVERVIEW

Raspberry Home is a secure private network solution designed to connect and manage small IoT devices efficiently within a local environment. It leverages a Raspberry Pi as a Layer 3 gateway to create an isolated network that enhances device security by preventing direct internet exposure [3]. The system integrates secure Over-the-Air (OTA) firmware updates, encrypted communication protocols, and remote access capabilities to provide a robust IoT management platform.

Types of IoT Private Networks:

IoT private networks vary in scale and security features depending on their intended applications. They range from simple local networks for basic device control to complex, enterprise-level architectures with multi-layered security and cloud integration.

Basic IoT Private Networks:

Basic networks provide local connectivity for IoT devices, often without advanced security. They enable fundamental functions such as data exchange and device control within a confined space, suitable for home automation or small projects.

Advanced Secure IoT Networks:

Advanced networks, like Raspberry Home, incorporate layered security measures including encrypted MQTT communication, secure OTA updates, and firewall protection. These networks support real-time monitoring, remote management through VPNs, and enhanced device authentication, ensuring privacy and resilience against cyber threats [4].

Key Components of Raspberry Home:

- **Layer 3 Gateway (Raspberry Pi):** Acts as the network's core router, managing traffic, device isolation, and enforcing security policies.
- **Secure OTA Updates:** Allows safe firmware upgrades over HTTPS, maintaining device software integrity.
- **Encrypted MQTT Communication:** Ensures data confidentiality and integrity between devices and the broker.

- **Firewall and Network Security:** Implements iptables and UFW rules to block unauthorized access and mitigate attacks.
- **Tailscale VPN:** Provides secure remote access for monitoring and managing devices without exposing the network publicly.

Advantages and Benefits of Raspberry Home:

1. Enhances IoT device security by isolating them from direct internet exposure.
2. Facilitates safe, version-controlled firmware updates, minimizing vulnerabilities.
3. Protects data exchange through strong encryption, reducing risks of interception.
4. Enables secure and convenient remote network access without complex configurations.
5. Provides a scalable, manageable framework ideal for home labs, educational purposes, and small-scale IoT deployments.

1.2 Objective

The primary aim of this thesis is to develop a comprehensive and secure networking solution tailored for small IoT devices, emphasizing autonomy, privacy, and ease of management. The specific goals include:

- **Creating a private network environment** that isolates IoT devices from the internet to enhance security.
- **Implementing secure Over-the-Air (OTA) firmware update mechanisms** to allow remote and controlled device management.
- **Ensuring data confidentiality and integrity** via TLS encryption for communication between IoT devices and the message broker.
- **Establishing robust network security controls** through firewall configurations using iptables and Uncomplicated Firewall (UFW).
- **Providing secure remote access** to the IoT network through Tailscale VPN, enabling authenticated users to monitor and manage devices without exposing the network publicly.

1.3 PROBLEM STATEMENT

The integration of IoT devices into everyday environments introduces significant cybersecurity and privacy challenges, including exposure to internet-based attacks, lack of secure firmware updates, unencrypted data transmission, insecure remote access methods, and the hardware limitations of low-cost microcontrollers like the ESP8266, which hinder the implementation of robust security protocols.

1.4 THESIS STRUCTURE

This thesis is organized into six comprehensive chapters, each addressing a key aspect of the Raspberry Home project and its approach to solving common IoT security issues.

Chapter 1: Introduction

This chapter introduces the Raspberry Home project by contextualizing the growing adoption of IoT devices in modern environments and the accompanying rise in cybersecurity and privacy threats. It outlines the key challenges faced in current IoT deployments, such as insecure internet exposure, lack of encrypted communication, and weak firmware update mechanisms.

Chapter 2: Literature Review

This chapter provides a detailed review of existing solutions related to IoT security, including communication protocols (e.g., MQTT, CoAP), firmware update strategies (OTA), and remote access approaches (VPNs, port forwarding

Chapter 3: System Architecture

This chapter details the overall hardware and network architecture of the Raspberry Home system. It explains the rationale for selecting devices like the Raspberry Pi (as a Layer 3 gateway) and ESP8266/ESP32 microcontrollers. The network topology is outlined, including the use of DHCP, firewall rules, and interface bridging to create an isolated and controlled IoT subnet.

Chapter 4: Software Implementation

This chapter focuses on the software-side configuration and deployment. It walks through the installation of key services on the Raspberry Pi, including the Mosquitto MQTT broker with TLS, Apache HTTP server for OTA file hosting, and Tailscale for secure remote access.

Chapter 5: Results and Evaluation

This chapter presents the functional and security testing outcomes of the Raspberry Home system. Tests include successful communication between nodes, secure firmware updates, and validation of encrypted data transmission.

Chapter 6: Conclusion and Future Work

The final chapter summarizes the key achievements of the Raspberry Home project, emphasizing its contribution to secure, private, and efficient IoT networking. It reflects on the project's ability to address core security gaps without relying on cloud platforms or expensive hardware

1.5 CONCLUSION

This chapter has outlined the motivation and objectives behind the Raspberry Home project, emphasizing the importance of security, privacy, and manageability in IoT networks. By leveraging a Raspberry Pi as a Layer 3 gateway and integrating secure OTA updates, encrypted communication, firewall protections, and VPN-based remote access, this project addresses critical challenges faced by IoT deployments today. The following chapters will delve deeper into the theoretical background, system design, practical implementation, and comprehensive evaluation of the proposed solution.

CHAPTER 2

LITERATURE SURVEY

2.1 INTRODUCTION

The rise of the Internet of Things (IoT) has transformed modern connectivity, enabling intelligent interaction between devices and systems. However, this rapid integration also introduces significant security and privacy concerns, especially in low-cost, resource-constrained environments. This literature survey delves into the evolving landscape of secure IoT architectures, with a focus on private networking, encrypted communication, and secure firmware updates. It critically examines existing frameworks, technologies, and implementations to assess how current solutions address these security challenges. By identifying limitations in conventional approaches, this review establishes the foundation for the Raspberry Home project, which aims to deliver a lightweight, practical, and secure IoT network for small-scale deployments.

2.1.1 MOTIVATION

The rapid expansion of IoT technology has revolutionized device interconnectivity, yet it has also exposed critical gaps in security and privacy, especially for small-scale and cost-sensitive deployments. Many off-the-shelf IoT devices prioritize convenience over protection, often lacking basic safeguards such as encrypted communication, secure firmware updates, or robust network isolation. This imbalance creates significant vulnerabilities that malicious actors can exploit, leading to data breaches, unauthorized control, and service disruptions.

Early implementations of IoT networks were typically open, relying on direct cloud connectivity and public IP exposure to enable remote access. While this approach simplified user interaction, it also introduced substantial risks due to weak or absent authentication, unencrypted data transmission, and lack of centralized control. As threats became more sophisticated, the need for secure, private, and manageable IoT ecosystems became evident.

The Raspberry Home project is motivated by this urgent need for a more secure and controlled IoT environment. It seeks to demonstrate how affordable components like Raspberry Pi and ESP microcontrollers can be leveraged to build a closed, encrypted, and remotely accessible network that defends against common IoT threats.

2.2 Synopsis of Previous Work

Research Paper 1: "Design and Implementation of House: An IoT Security Framework"

-Wen Fei, Hiroyuki Ohno, and Srinivas Sampalli, 2021

This paper introduces Raspberry Home, a TCP/IP Layer 3 gateway developed using a Raspberry Pi to provide enhanced security for small IoT devices. The system was designed to address the lack of secure networking environments for resource-constrained IoT hardware such as the ESP8266. The Raspberry Home architecture creates a private IP network, isolating connected devices from the internet while enabling secure communication, remote firmware updates, and encrypted data transmission.

The authors emphasize the system's real-world relevance for DIY developers and researchers using small development boards without built-in enterprise-grade security features. The implementation prioritizes isolation, accessibility, and modular integration of encryption and access control mechanisms.

Summary of Findings

The system was tested successfully under real-world scenarios, with the following key features:

1. Raspberry Pi as Layer 3 Gateway: Functions as a DHCP, DNS, and access point server using dnsmasq and hostapd, assigning private IPs and blocking internet access for internal devices.
2. OTA Firmware Updates: Secure firmware updates are enabled using FOTA. Binary files are uploaded via SCP, then flashed using espota.py, reducing physical interaction.
3. TLS Encryption: MQTT communication is protected using self-signed certificates and port 8883 configuration. While client authentication is supported, hardware limitations of devices like ESP8266 may restrict full mutual TLS.
4. SSH Port Forwarding: Remote access to internal systems is achieved using secure SSH tunnels, avoiding the risks of port forwarding and public IP exposure.
5. Firewall & Security Hardening: UFW, fail2ban, SSH key-based login, and system hardening steps are applied to enhance gateway security.

Limitations

The authors acknowledge certain limitations:

- Manual Updates: Raspberry Home lacks automated update features found in commercial IoT gateways, requiring manual configuration.
- ESP8266 Memory Constraints: Full certificate verification and mutual TLS authentication are not feasible on constrained devices like ESP8266.
- Reliance on Local Infrastructure: System assumes access to a local PC or server for file management and OTA deployment.

Despite these drawbacks, Raspberry Home offers significant security improvements over traditional plug-and-play cloud-based IoT solutions.

Research Paper 2: *OTA Measurement for IoT Wireless Device Performance Evaluation: Challenges and Solutions*

Penghui Shen, Yihong Qi, Wei Yu, Jun Fan, and Fuhai Li, IEEE Internet of Things Journal, 2022

This paper provides an in-depth study of Over-the-Air (OTA) testing methodologies used to evaluate the performance of IoT wireless devices. It emphasizes the growing importance of OTA evaluations for reliable communication in IoT systems. The study highlights that conventional cellular OTA standards are inadequate for IoT due to cost, speed, and accuracy limitations, especially with low-cost SISO and MIMO devices. To overcome these, the authors propose enhanced OTA methodologies tailored to constrained IoT environments.

The proposed techniques include rapid testing frameworks for SISO devices and an innovative Radiated Two-Stage (RTS) method for MIMO evaluation, providing significant improvements in accuracy and efficiency over traditional anechoic chamber testing.

Summary of Findings

Key contributions of the paper include:

1. Enhanced TIS Measurement Techniques: Introduces three fast methods for Total Isotropic Sensitivity (TIS) testing—RSSI-based, co-frequency, and BER curve-based—that reduce time and increase accuracy.
2. RTS Method for MIMO Devices: Replaces complex hardware setups (like MPAC) with baseband-simulated channels, enabling low-cost and accurate throughput testing in smaller chambers.

3. Smart Test Chamber Design: Proposes a compact, office-friendly OTA test environment supporting both SISO and MIMO evaluations.

4. Hardware Optimization: Highlights the role of antenna gain, PA output, and RF front-end

Limitations

- Focused on Physical Layer Testing: Does not evaluate full-stack performance, such as encryption or protocol behavior under OTA conditions.
- Requires Sophisticated Modeling: Accurate simulation for RTS in MIMO requires detailed channel modeling, which can be computationally intensive.
- Standardization in Progress: While accepted by 3GPP, RTS still lacks widespread industry adoption and implementation tools.

Research Paper 3: *Secure Firmware Updates for Constrained IoT Devices Using Open Standards: A Reality Check*

Koen Zandberg, Kaspar Schleiser, Francisco Acosta, Hannes Tschofenig, Emmanuel Baccelli, IEEE Access, 2019

This paper addresses one of the most critical challenges in IoT security—secure firmware updates for constrained devices. It presents a prototype system built entirely on open standards such as IETF SUIT, CoAP, and LwM2M. These tools enable secure, authenticated firmware delivery without relying on proprietary solutions, making them ideal for low-power devices with limited memory and compute resources.

The authors evaluate several update mechanisms, including SUIT-manifest-based updates, and demonstrate that secure OTA can be achieved even on devices with under 32 KB RAM and 128 KB flash.

Summary of Findings

Major features and insights from the prototype include:

1. Standards-Based Update Architecture: Uses IETF SUIT manifests, CoAP transport, and digital signatures (ed25519/ECDSA) for verifying firmware authenticity.

1. Flexible Configurations: Offers multiple update modes—Basic-OTA, IPv6-OTA, SUIT-OTA, and LwM2M—with options for push/pull updates and secure metadata handling.

2. Lightweight Design: Implemented on RIOT OS, the firmware update module supports buffer-based flash writing and signature verification in low-resource environments.

3. Lifecycle Security: Supports trust anchor management, secure delegation, and crypto agility for future-proofing firmware integrity.

Limitations:

- No Differential Updates: Only full firmware images are supported, increasing bandwidth and energy consumption.

- Single Trust Anchor Model: A compromised private key breaks the system unless key rotation mechanisms are added.

- No Built-in TLS: While CoAP is used, the system avoids DTLS for simplicity, which may limit applicability in sensitive environments.

2.3 Gap Analysis and Research

Analyzing the literature surrounding IoT-based home automation, secure gateway systems, OTA firmware updates, and wireless communication performance reveals several key gaps that Raspberry Home aims to address.

1. Lack of Network Isolation in Home IoT Setups

Most existing home automation systems—such as the one proposed by Fei Hu—focus on functionality and ease of access but do not prioritize network segmentation or device isolation. IoT devices are often connected to the same network as user devices, increasing exposure to cyber threats.

Gap: Absence of secure network boundaries between IoT devices and the internet. **Research Focus**: Raspberry Home addresses this by establishing a Layer 3 gateway using Raspberry Pi and implementing network isolation with private IP assignment, thereby reducing attack surface and improving overall security.

2. Insecure Firmware Update Mechanisms

Studies like that of Zandberg et al. highlighted that constrained devices often lack secure OTA infrastructure. Full firmware images are delivered without encryption, and many systems lack verification or rollback options.

Gap: Firmware updates are not securely transmitted or verified, making devices vulnerable to tampering.

Research Focus: Raspberry Home enables HTTPS-based OTA updates hosted on an Apache server with version control and remote firmware management, ensuring secure and authenticated firmware delivery.

3. **Limited Use of Encrypted Communication Protocols**

Hu's system and similar lightweight IoT setups often skip secure communication protocols due to complexity or performance trade-offs. This exposes data-in-transit to eavesdropping and injection attacks.

Gap: Lack of TLS-encrypted communication in MQTT or CoAP-based messaging systems.

Research Focus: Raspberry Home integrates TLS encryption for MQTT communication. Although full mutual TLS is constrained by ESP8266 hardware, the system still ensures one-way authentication and encrypted data transfer.

4. **Complex and Unsafe Remote Access Techniques**

Many existing systems rely on port forwarding or static public IPs to allow remote access to the network. This approach introduces severe security vulnerabilities by exposing the gateway to the public internet.

Gap: Unsafe remote access configurations using port forwarding or weak VPNs.

Research Focus: Raspberry Home uses Tailscale VPN, a peer-to-peer, zero-trust overlay network that allows encrypted remote access without exposing the Raspberry Pi or devices to the internet.

5. **Insufficient OTA Performance Metrics and Wireless Reliability**

While the OTA evaluation paper by Penghui Shen et al. introduces improved testing methods, it also highlights that many DIY and low-cost IoT systems lack performance metrics such as signal quality and update reliability.

Gap: No OTA performance optimization or RF reliability testing in small-scale home deployments. **Research Focus:** While Raspberry Home does not include formal RF test infrastructure, it follows best practices such as OTA stability testing, log monitoring, and ensures communication resilience via Wi-Fi signal tuning and OTA retries.

6. **Absence of Integrated Firewall and Localized Security Policies**

Few IoT gateway solutions implement firewall rules for traffic control between IoT and management interfaces. The lack of local security enforcement allows malware or unauthorized traffic to spread within the network.

Gap: No device-level traffic control or firewall integration in existing systems. **Research Focus**: Raspberry Home incorporates a firewall using iptables and UFW, which restricts access between network interfaces, enforces DHCP range control, and protects exposed services on the Raspberry Pi.

2.4 Conclusion

To address these gaps, Raspberry Home proposes a multi-layered, security-first architecture with the following contributions:

- Layer 3 Gateway Design: Isolates IoT traffic using a custom subnet, ensuring devices cannot reach the internet directly or interact with unauthorized hosts.
- Secure OTA Updates: OTA firmware delivery over HTTPS with versioning and remote file upload via SCP or WinSCP.
- TLS-Encrypted MQTT Communication: Ensures data confidentiality even on constrained hardware (ESP8266), with future support planned for mutual TLS on ESP32.
- Tailscale VPN Integration: Provides seamless, encrypted, and identity-aware remote access without using public IPs or NAT traversal.

CHAPTER 3

SYSTEM HARDWARE AND NETWORK DESIGN

3.1 INTRODUCTION

This project aims to design RASPBERRY HOME – SECURE PRIVATE NETWORK FOR IoT DEVICES, addressing the growing need for secure and manageable IoT environments in homes and small-scale deployments. The system uses a Raspberry Pi as a Layer 3 gateway to isolate connected IoT devices, primarily ESP8266 and ESP32 modules, within a private network.

The design incorporates multiple components including the Raspberry Pi acting as the network gateway and server, an Apache web server for secure OTA firmware updates, a Mosquitto MQTT broker for encrypted communication, and firewall tools such as iptables and UFW for network security. Additionally, Tailscale VPN is integrated for safe remote access, ensuring controlled and encrypted connectivity without exposing the devices to the public internet[1]a.

This section details the hardware and software components used, along with their specifications and roles in achieving a robust, privacy-focused IoT network.

3.2 HARDWARE COMPONENTS

3.2.1 Raspberry Pi 3B

The Raspberry Pi 3 Model B is a powerful single-board computer widely used in IoT and embedded projects. It features a Broadcom BCM2837 SoC with a 64-bit quad-core ARM Cortex-A53 processor running at 1.2 GHz. This model includes onboard Wi-Fi and Bluetooth capabilities, making it highly suitable for networking and IoT applications.

The Raspberry Pi 3 Model B supports various interfaces and peripherals, allowing it to act as a network gateway, server, and controller in projects like the Raspberry Home secure IoT network.

Specifications & Features of Raspberry Pi:

1. **Processor:** Broadcom BCM2837, Quad-Core ARM Cortex-A53, 64-bit, 1.2 GHz
2. **Memory (RAM):** 1 GB LPDDR2 SDRAM
3. **Operating Voltage:** 5V via Micro USB power supply
4. **Storage:** microSD card slot for OS and data storage
5. **Networking:**
6. 802.11 b/g/n Wireless LAN (Wi-Fi)
7. Bluetooth 4.1 Classic and Bluetooth Low Energy (BLE)
8. 10/100 Mbps Ethernet port
9. **USB Ports:** 4 × USB 2.0 ports
10. **GPIO Pins:** 40-pin GPIO header (fully backward-compatible with previous models)
11. **Video Output:** HDMI port supporting 1080p video output
12. **Audio Output:** 3.5mm analog audio jack and HDMI audio
13. **Camera Interface (CSI):** 15-pin MIPI camera serial interface
14. **Display Interface (DSI):** 15-pin MIPI display serial interface
15. **Power Management:** Supports Power over Ethernet (PoE) with add-on module
16. **Size:** 85.6 mm × 56.5 mm

The 40-pin GPIO header includes multiple general-purpose input/output pins, I2C, SPI, UART interfaces, and power pins (3.3V, 5V, and ground), providing versatile options for connecting sensors, actuators, and other peripherals.

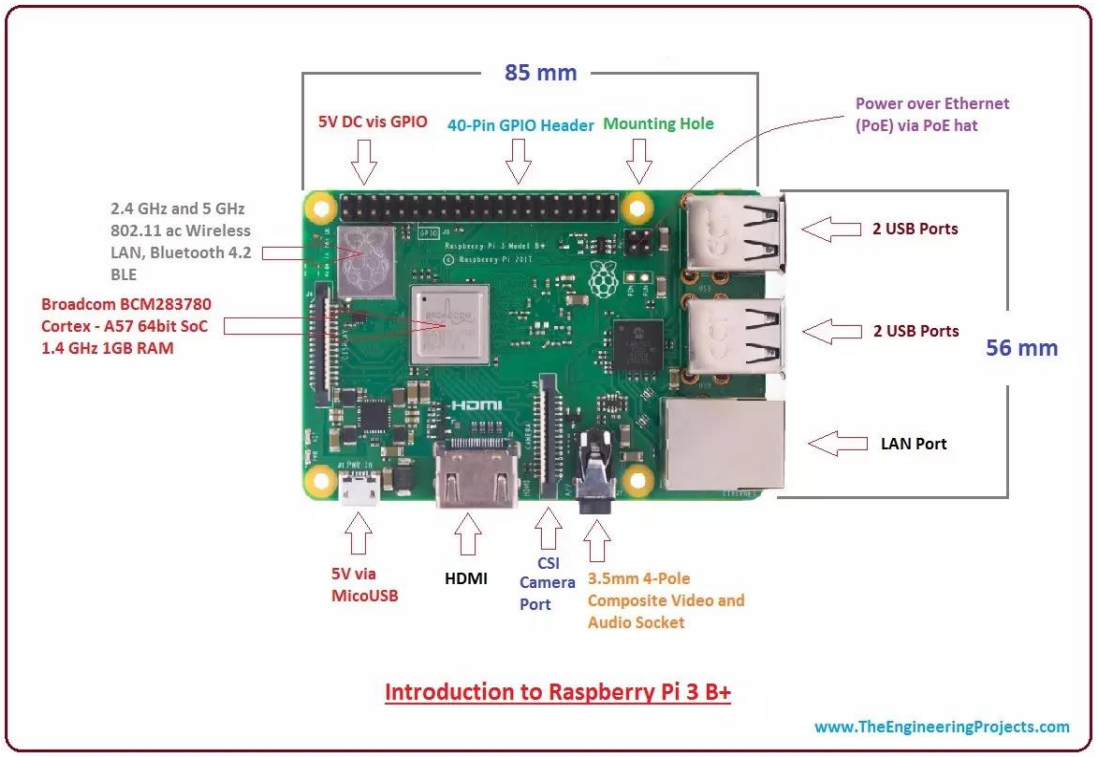

Fig 3.1: Figure shows front view of raspberry pi 3B

The Raspberry Pi 3 Model B serves as the core Layer 3 gateway device in the Raspberry Home project. Its role is to create and manage a secure private network for IoT devices, provide encrypted communication channels, host OTA update servers, and enable secure remote access.

Key features used in this project:

Processor & Performance:

- The BCM2837 SoC with a 1.2 GHz quad-core ARM Cortex-A53 processor provides sufficient computing power to run the Linux-based operating system (Raspbian) and network services like Mosquitto MQTT broker, Apache web server for OTA updates, and firewall management tools without latency issues.

Networking:

- **Built-in Wi-Fi (802.11 b/g/n):** Used to create the wireless private IoT network access point. This enables IoT devices like ESP32 and ESP8266 to connect securely without relying on external routers.
- **Ethernet port:** Used for connecting the Raspberry Pi to the broader home or lab network for internet access and remote management when needed.
- **Security Features in Use:**
- TLS encryption for MQTT communication to secure sensor data in transit.
- HTTPS hosting of OTA firmware updates via Apache server, ensuring secure and authenticated update delivery.
- Firewall (iptables and UFW) configured to restrict network access only to authorized devices.
- Remote access via Tailscale VPN, providing encrypted, peer-to-peer connections without exposing the Raspberry Pi directly to the internet.

Ethernet Port:

- The Ethernet port connects the Raspberry Pi to the **existing home or lab network** that has internet access.[6]
- It acts as the **uplink interface**, receiving internet connectivity from the broader network or router.
- The Raspberry Pi acts as a **gateway** — routing and managing traffic between the IoT devices on the private wireless network and the external internet via this Ethernet connection.
- Through this setup, the Pi can also reach external servers for tasks like remote updates or VPN connections without exposing the IoT devices directly to the internet.

Wi-Fi Interface (Configured as Wireless Access Point - WAP):

- The built-in Wi-Fi module is configured to function as a **wireless access point (WAP)**.
- This access point **broadcasts a private wireless network** exclusively for your IoT devices (e.g., ESP32, ESP8266).
- IoT devices connect to this private Wi-Fi network, isolating them from the main home network to enhance security.
- The Raspberry Pi manages IP addressing (using a DHCP server) and network traffic within this isolated subnet.

- The Pi controls which devices can connect and communicate, enforcing firewall rules to prevent unauthorized access or exposure.
- This WAP configuration allows seamless communication between IoT devices and the gateway for MQTT messaging, OTA updates, and other control functions.

Pin Description of Raspberry Pi

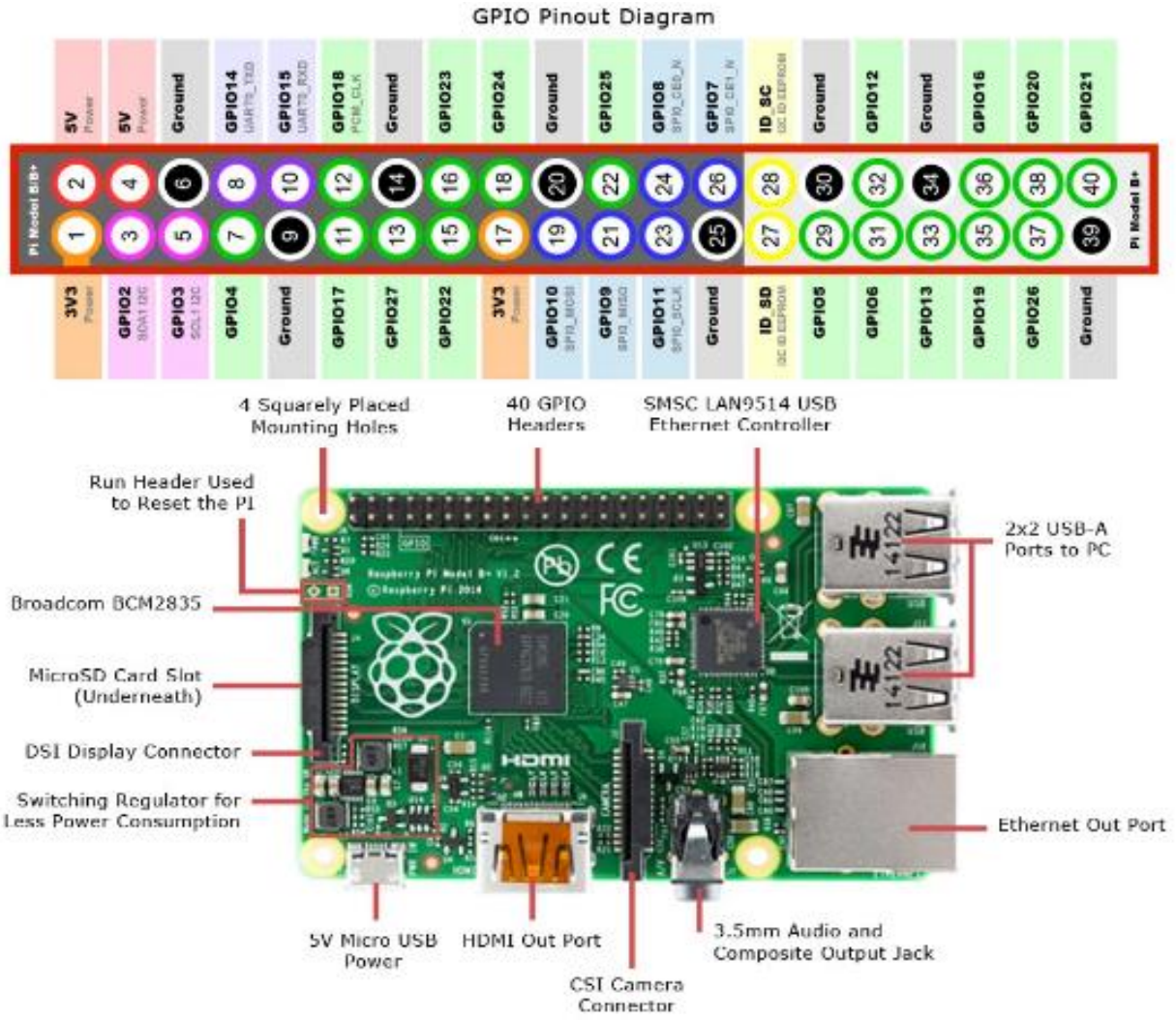

Figure 3.2 Figure Shows the pin Out diagram of Raspberry Pi 3B

3.2.2 Nodemcu

NodeMCU is an open-source IoT platform. It includes firmware which runs on the ESP8266 Wi-Fi SoC from Espressif Systems, and hardware which is based on the ESP-12 module. The term "NodeMCU" by default refers to the firmware rather than the dev kits. The firmware uses the Lua scripting language. It is based on the eLua project and built on the Espressif Non-OS SDK for ESP8266. It uses many open-source projects, such as lua-cjson and spiffs. [9]

NodeMCU has **128 KB RAM and 4MB of Flash memory** to store data and programs. Its high processing power with in-built Wi-Fi / Bluetooth and Deep
Sleep Operating features make it ideal for IoT projects. NodeMCU can be powered using a Micro USB jack and VIN pin (External Supply Pin) as shown in figure 3.3.

NodeMCU ESP8266 – Specifications & Features

1. **Microcontroller:** Tensilica 32-bit RISC CPU Xtensa LX106
2. **Operating Voltage:** 3.3V
3. **Input Voltage:** 7–12V
4. **Digital I/O Pins (DIO):** 16
5. **Analog Input Pins (ADC):** 1
6. **UARTs:** 1
7. **SPIs:** 1
8. **I2Cs:** 1
9. **Flash Memory:** 4 MB
10. **SRAM:** 64 KB
11. **Clock Speed:** 80 MHz
12. **USB-TTL Converter:** Onboard CP2102 chip for Plug and Play functionality
13. **Antenna:** Integrated PCB Antenna
14. **Form Factor:** Compact size ideal for embedding in IoT projects

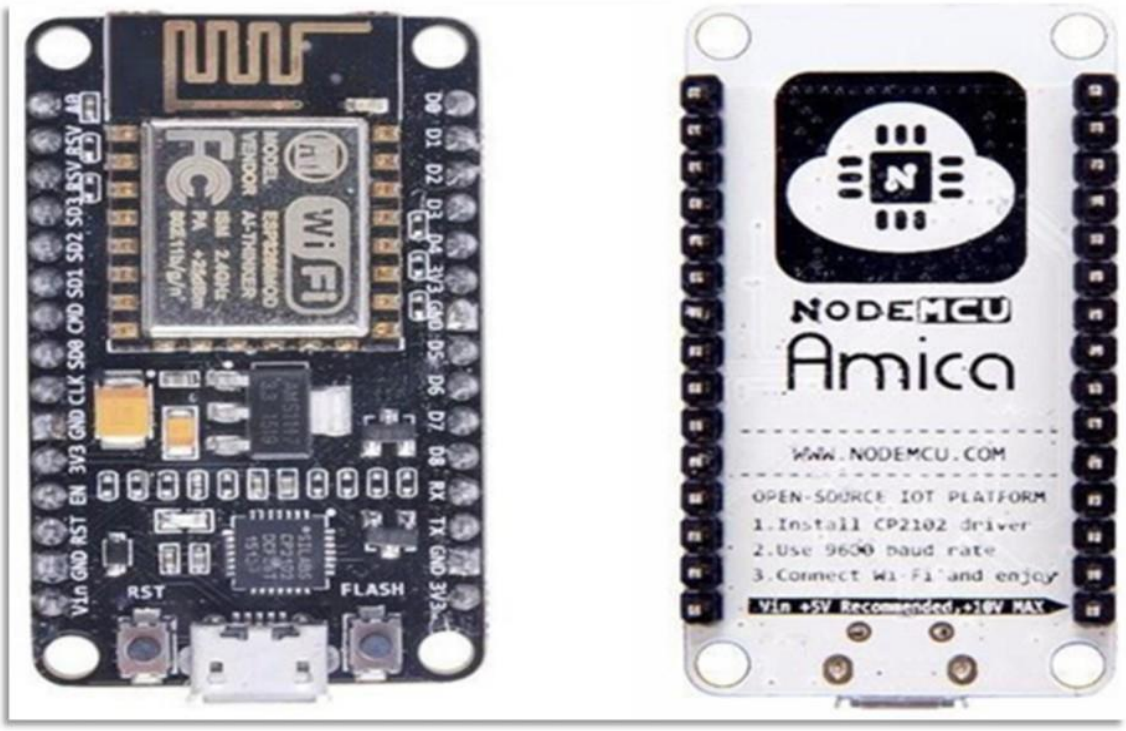

Figure 3. 3 Figure Shows Nodemcu Front and Back view

ESP8266 CHIP

The ESP8266 is a low-cost Wi-Fi chip with full TCP/IP stack and MCU (Micro Controller Unit) Capability produced by shanghai-based chinese manufacturer, Espressif systems The chip first The chip first came to the attention of western makers in August 2014.

Figure 3.4: Figure Shows Nodemcu Module

However, at the time there was almost no English-language documentation on the chip and the commands it accepted. The very low price and the fact that there were very few external components on the module which suggests that it could eventually be very inexpensive in volume, attracted many hackers to explore the module, chip, and the software on it, as well as to translate the Chinese documentation.

NodeMCU Pinout

For practical purposes **ESP8266 NodeMCU** V2 and V3 boards present identical pinouts. While working on the NodeMCU based projects we are interested in the following pins.

Power pins (3.3 V). Ground pins (GND). Analog pins (A0).

Digital pins (D0 – D8, SD2, SD3, RX, and TX – GPIO XX)

Most ESP8266 NodeMCU boards have one input voltage pin (Vin), three power pins (3.3v), four ground pins (GND), one analog pin (A0), and several digital pins (GPIO XX).

Figure 3.5 shows the pinout for the NodeMCU board. A typical NodeMCU board (if it is based on the original NodeMCU Devkit design) has 30 pins. In this, 8 pins are related to power and 2 are reserved. The remaining 20 pins are associated with pins of the ESP-12E [7]

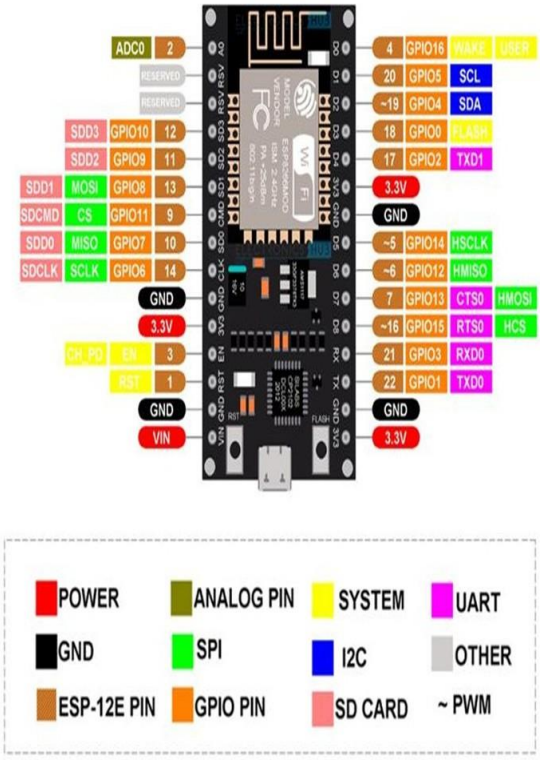

Figure 3. 5 Figure Shows Nodemcu pinout

3.2.3 IoT Device 1: Temperature and Humidity Sensor with ESP8266

This device is a crucial environmental monitoring component within the Raspberry Home ecosystem. It continuously measures temperature, and humidity levels and sends the data wirelessly to the central Raspberry Pi gateway. This information can be used for real-time monitoring, logging environmental changes, and triggering automated responses, such as turning on ventilation or alerts if certain thresholds are exceeded.

Hardware Components:

- ESP8266 Wi-Fi Module: The ESP8266 serves as the microcontroller unit (MCU) in this device. It manages sensor data acquisition and communication over Wi-Fi. The ESP8266 is selected for its integrated Wi-Fi capability, low cost, and sufficient processing power to handle sensor interfacing and MQTT communication.
- DHT11 Sensor: This is a popular and reliable digital sensor that measures both temperature and humidity. It communicates with the ESP8266 using a single-wire digital protocol, making it easy to interface. The DHT11 provides calibrated, relatively accurate data for typical indoor environmental monitoring, with a temperature range from -40°C to 80°C and humidity range from 0% to 100%.
- Power Supply: Typically powered at 3.3V, from a regulated DC power source or battery, optimized for low power consumption to enable longer deployment without frequent recharging.

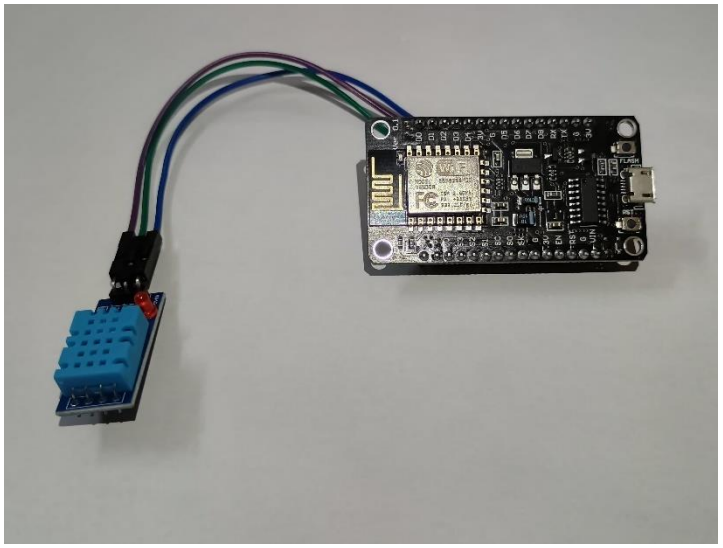

Figure 3.6 Figure Shows IoT device 1

Table 3.1: DHT11 Temperature and Humidity Sensor Specifications

Parameter	Specification
Sensor Type	Digital Temperature and Humidity Sensor
Temperature Range	0°C to 50°C
Temperature Accuracy	±2°C
Humidity Range	20% to 90% RH
Humidity Accuracy	±5% RH
Operating Voltage	3.3V to 5V
Current Consumption	0.5 mA during measurement
Sampling Rate	1 Hz (once per second)
Signal Output	Digital signal via single-wire protocol
Response Time	6 seconds
Interface Type	Single-wire digital interface
Dimensions	Approximately 15.5mm x 12mm x 5.5mm

Functionality and Operation:

- The DHT22 sensor periodically samples the ambient temperature and relative humidity.
- The sensor transmits the data to the ESP8266 MCU through its single-wire digital data line.
- The ESP8266 processes this raw data, formats it into JSON or a similar structured message, and publishes it to the MQTT broker hosted on the Raspberry Pi.
- Communication is secured using TLS encryption, protecting sensor data from interception or tampering within the private network.
- The ESP8266 firmware includes logic to put the device into deep sleep mode between readings, significantly conserving battery life if deployed wirelessly.

3.2.4 IoT Device 2: Smart Switch Using ESP8266, 5V Relay, and Connected Bulb

This device functions as a remotely controllable smart switch capable of toggling an electrical appliance—such as a light bulb—on or off from anywhere on the private network. It exemplifies how traditional electrical devices can be integrated into a smart home environment with secure, reliable wireless control.

Hardware Components:

- ESP8266 Wi-Fi Module: Acts as the main controller, receiving control commands over Wi-Fi and driving the relay accordingly.
- 5V Relay Module: Electrically isolates and switches the connected electrical load (bulb) safely. The relay coil is powered separately from the control signal to protect the MCU from voltage spikes and surges.
- Electrical Load (Bulb): The controlled device, which can be any appliance compatible with the relay's voltage and current ratings.
- Power Supply: The ESP8266 runs on 3.3V, whereas the relay coil typically requires a stable 5V supply, often provided by a separate regulated source or through the Raspberry Pi's 5V pin.

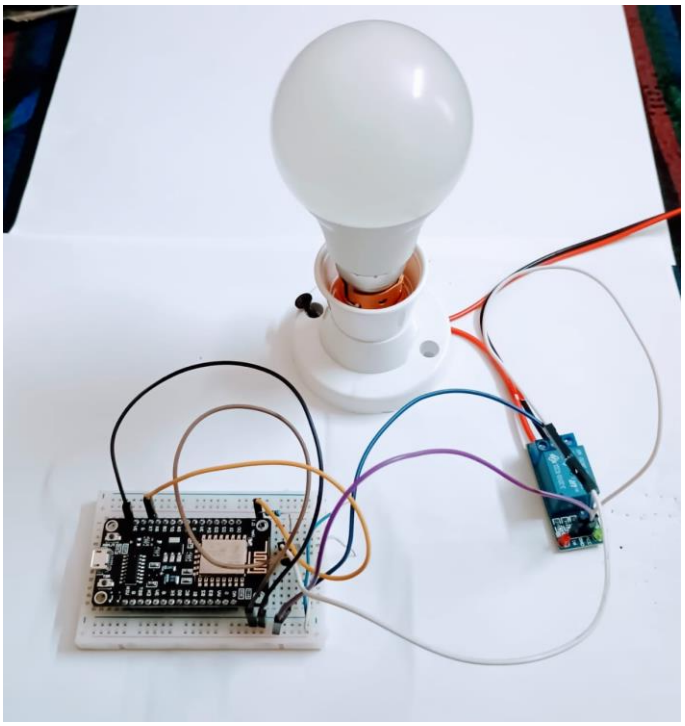

Fig 3.7: Figure Shows IoT device

Table 3.2:5V Relay Module Specifications

Parameter	Specification
Operating Voltage	5V DC (coil voltage)
Control Signal Voltage	3.3V to 5V (compatible with ESP8266 GPIO)
Contact Rating	10A at 250VAC / 10A at 30VDC
Relay Type	Electromechanical, Single Pole Double Throw (SPDT)
Isolation	Optocoupler isolation between control and relay coil
Trigger Type	Active low or active high (depends on module design)
Input Current	~70mA (coil current)
Switching Time	Operate time: ~10ms, Release time: ~5ms
Indicators	LED indicator for relay activation

Functionality and Operation:

- The ESP8266 subscribes to a specific MQTT topic on the Raspberry Pi broker, listening for control messages such as “ON” or “OFF.”
- Upon receiving a command, the ESP8266 sets the GPIO pin connected to the relay module to HIGH or LOW, energizing or de-energizing the relay coil.
- The relay contacts switch the electrical circuit of the bulb, turning it on or off without direct electrical contact with the ESP8266, ensuring device and user safety.
- Status feedback is also published back to the broker, enabling real-time monitoring of the switch state via dashboards or mobile apps.
- The relay module typically includes an optocoupler to provide galvanic isolation between the high voltage side and the ESP8266 control circuitry[9] .

3.3 Network Architecture of Raspberry Home

Overview

The network architecture of Raspberry Home is designed with a strong focus on security, isolation, and manageability. At its core, a Raspberry Pi 3 Model B functions as a **Layer 3 gateway** and **secure service host**, bridging two distinct network domains:

- A **private IoT subnet** where ESP-based devices communicate securely without direct internet access.

- An **uplink interface** connected to the wider home or lab network (via Ethernet), which provides internet connectivity for the gateway itself and enables remote access through Tail scale VPN.

This dual-interface configuration allows the Raspberry Pi to enforce strict control over routing, DNS, DHCP, and firewall policies between the internal IoT network and external access points.

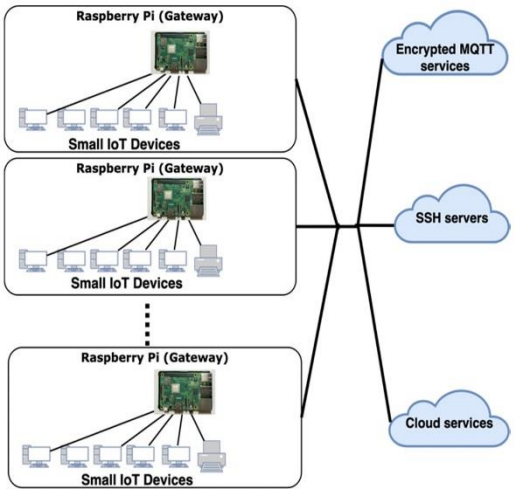

Fig 3.8: Figure Shows Implementation Framework

1.Physical Network Setup

- **Raspberry Pi 3B**:
- **Wi-Fi Interface (wlan0):** Configured as a **Wireless Access Point (WAP)** using hostapd, it creates and broadcasts the **IoT network SSID**.
- **Ethernet Interface (eth0):** Connectd to the main router or LAN to access internet services, perform remote updates, or establish a VPN tunnel.
- **ESP8266 / ESP32 Devices**:

- Connect to the **private IoT Wi-Fi** broadcasted by the Pi.
- Communicate over **TLS-secured MQTT** and perform **OTA updates over HTTPS**.
- **Administrator Laptop / Smartphone**:
- Can access the Pi either:
- Locally (via LAN),
- Via **Tailscale VPN** from any location,
- Or temporarily via SSH when on the same LAN.

2. Logical Network Architecture

The architecture can be viewed as a **dual-subnet system** managed entirely by the Raspberry Pi:

A. Internal IoT Network (wlan0 – AP Mode)

- **Subnet Range:** 192.168.50.0/24
- **Services provided by the Pi:**
    - **DHCP** using dnsmasq
    - **DNS resolution** for internal devices
    - **NAT** with iptables (if temporary internet access is granted)
    - **Firewall** via UFW and iptables
    - **MQTT Broker** (Mosquitto with TLS on port 8883)
    - **Apache2 Web Server** for OTA updates (https://raspberrypi.local/update/firmware.bin)
- **Isolation Measures:**
    - ESP devices **cannot reach the internet**
    - No communication between IoT devices unless explicitly allowed
    - All traffic is filtered and logged

B. Uplink to Home Network or Internet (eth0)

- **Subnet Range:** Typically, 192.168.1.0/24 or 10.0.0.0/24 (depends on home router)
- **Purpose:**
- Used only by the Pi for:
- Remote SSH administration (when allowed)
- Downloading packages/updates

3. Remote Access via Tailscale VPN

- **Tailscale** creates a **zero-trust, peer-to-peer network overlay** using WireGuard.
- Once authenticated, remote devices can securely access:
- The **MQTT broker** (for data monitoring or control),
- The **OTA update portal**,
- The **device logs** or **sensor dashboards** hosted locally.
- No **public IP**, **port forwarding**, or **dynamic DNS** is required.

4. MQTT Communication Flow

- Devices connect to mqtts://192.168.50.1:8883
- **TLS encryption** is used for data integrity and privacy
- Authentication is managed using usernames and passwords (stored in mosquitto_passwd), and optionally a server certificate
- ESP devices **publish sensor data** or **receive commands** securely

5. OTA Firmware Update Flow

- Firmware is uploaded manually to /var/www/html/firmware/ on the Pi via **SCP/WinSCP**
- ESP devices connect securely to:
- https://192.168.50.1/firmware/firmware.bin
- Devices check for version updates before flashing
- HTTPS ensures that firmware images are not tampered with during transmissio

6. Security Architecture

Table 3.3: Architecture Flow

Component	Security Implementation
MQTT Broker	TLS encryption, optional client authentication
OTA Updates	HTTPS hosting, version control, SCP upload
Remote Access	Tailscale VPN with identity-based access
Firewall	UFW and iptables-based interface-level rules
Device Access	Private subnet and NAT isolation
SSH Access	Key-based login only, fail2ban enabled

7. Block Diagram

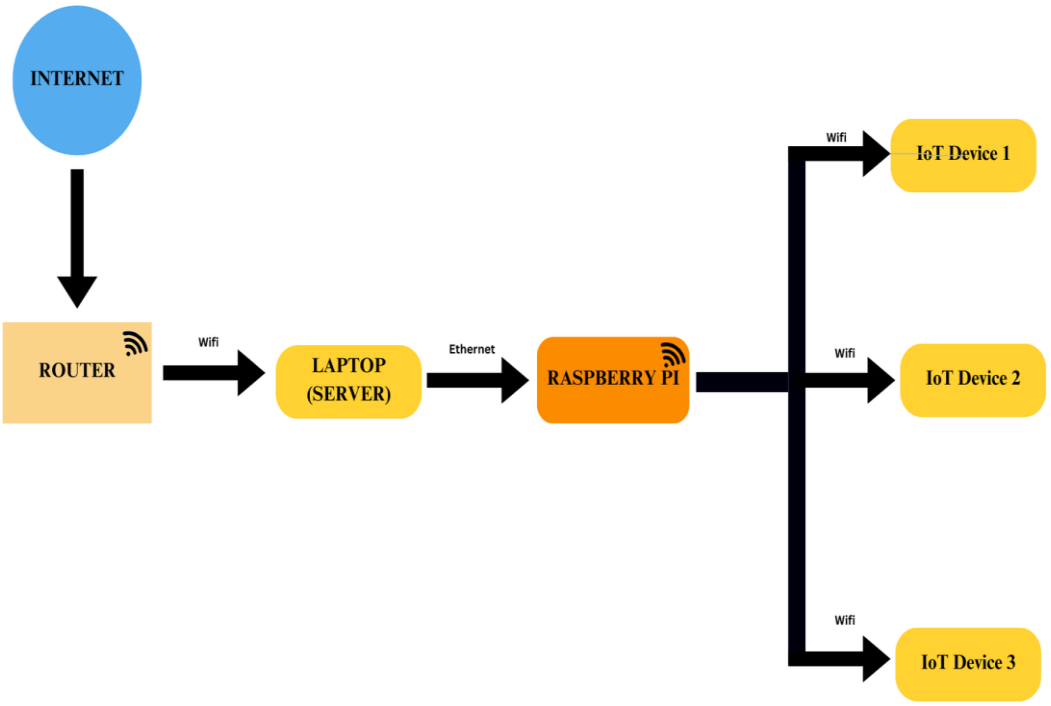

Figure 3.9 Figure Shows Block diagram of Rasp Home

3.4 Conclusion

The development of the Raspberry Home project demonstrates a robust, scalable, and security-focused approach to managing small IoT devices within an isolated, private network environment. The project integrates networking, embedded systems, security protocols, and remote access technologies into a cohesive solution tailored for constrained devices like ESP8266 and ESP32. The following points highlight the major outcomes and contributions of this work:

1. Secure Network Architecture: The Raspberry Pi was successfully configured as a Layer 3 gateway, broadcasting a private subnet and fully isolating IoT devices from the public internet. By integrating dnsmasq, hostapd, and iptables, the system enforces strict network boundaries, significantly reducing the risk of external threats.
2. Encrypted Communication: The project achieved secure MQTT messaging using TLS encryption, ensuring data integrity and confidentiality between IoT sensors, actuators, and the central broker. Despite the hardware limitations of the ESP8266, the system supports encrypted one-way

communication, with a roadmap for future mutual authentication using ESP32.

3. Secure OTA Firmware Updates: An Apache2 HTTPS server hosted on the Raspberry Pi delivers firmware updates securely to IoT nodes. Devices verify version changes and perform over-the-air updates without manual flashing, demonstrating efficient remote management. Firmware files are pushed via WinSCP, maintaining a controlled update pipeline.
4. Remote Access with Zero-Trust Principles: By integrating Tailscale VPN, the project enables seamless, identity-based remote access to the IoT network without exposing any ports or requiring a public IP. This eliminates the need for port forwarding, dramatically increasing the security posture of the system.
5. Firewall and Access Control: Using iptables and UFW, strict inbound and outbound traffic rules were enforced. IoT devices are restricted from reaching external networks, while administrative access is only permitted through defined secure channels.
6. Demonstrative IoT Devices: Two functional devices were developed:
   A temperature and humidity sensor that publishes environmental data securely over MQTT.

Future Work

- Full Mutual TLS Support: With a shift to ESP32 or other more capable devices, the system can be upgraded to support client-side certificate validation, enabling complete mutual TLS authentication for all MQTT communications.
- Web-Based Dashboard: Integrating a real-time monitoring dashboard on the Raspberry Pi would enhance visualization of sensor data, device statuses, and logs, offering a complete interface for managing the private IoT ecosystem.
- Automated Update Deployment: Implementing a backend service to track device versions and automatically deploy OTA updates based on rules or schedules would streamline firmware management further.
- Intrusion Detection and Logging: Extending the firewall to include logging, fail2ban integration, and anomaly detection could provide real-time alerts and insights into network behavior, further hardening the system.
- Support for Additional Protocols: Expanding the system to support CoAP or HTTPS-based REST APIs can open up compatibility with broader IoT ecosystems and commercial dashboards.

Final Thoughts

The Raspberry Home project illustrates the power of combining network engineering, cybersecurity, and embedded systems to build a secure and autonomous IoT infrastructure. Unlike typical plug-and-play IoT setups that prioritize ease over security, this project prioritizes privacy, control, and resilience through a tightly integrated, cloud-independent system.

From a systems design perspective, the project required deep integration of networking components (firewall, DNS, DHCP), secure communication layers (TLS, VPN), and hardware-level firmware delivery mechanisms (OTA). Each layer was carefully chosen and implemented to balance performance with the constraints of small IoT devices.

Educationally, the project provided valuable experience in real-world system integration, Linux server configuration, Wi-Fi network management, TLS certificates, and embedded firmware development.

CHAPTER 4
SOFTWARE STACK DESIGN AND INTEGRATION

4.1 Introduction

The integration of software tools plays a pivotal role in the functionality, security, and manageability of IoT systems. In the Raspberry Home project, software integration was foundational to creating a secure, private, and autonomous environment for connected IoT devices. This chapter explores the core software components, their roles, and how they were implemented to achieve the project's objectives. At the heart of the system is the Raspberry Pi, which acts as a Layer 3 gateway. It performs critical network functions such as routing, NAT, DHCP, and firewall enforcement. Key to this setup is the use of dnsmasq to serve DHCP and DNS services within a completely isolated subnet. The Pi also runs `iptables` in conjunction with Uncomplicated Firewall (UFW) to enforce strict ingress and egress rules, blocking all internet-bound traffic from the IoT subnet while allowing necessary internal communication. This ensures robust device isolation and limits exposure to external threats.

A secure Over-the-Air (OTA) update mechanism was implemented using an HTTPS-enabled Apache web server hosted on the Raspberry Pi. Firmware files are securely transferred using WinSCP and served over HTTPS. ESP8266/ESP32 devices periodically check for firmware version updates and, if available, download and install them autonomously. This was facilitated by embedded HTTP clients using the `ESP8266HTTPClient.h` and `WiFiClientSecure.h` libraries. Firmware integrity is ensured via TLS encryption, and version control logic prevents repeated or unauthorized updates.

For encrypted data communication, the Mosquitto MQTT broker was deployed on the Pi with TLS support. While client-server certificate verification was initially considered, it was not implemented due to memory limitations on the ESP8266. Nevertheless, data in transit is encrypted using TLS, offering confidentiality and a substantial layer of security for messages passed between devices and the broker.

Remote access to the network is achieved through the integration of Tailscale, a zero-config VPN built on WireGuard. Tailscale establishes a mesh VPN between trusted client devices and the Raspberry Pi, allowing remote control and monitoring of the private IoT network without the need for port forwarding or public IP addresses. Its peer-to-peer architecture and device-level authentication offer a secure and intuitive way to manage the system remotely. Each software tool in Raspberry Home has been carefully selected and configured to address specific challenges in IoT system security and remote management. Together, they form a cohesive software stack that ensures privacy, resilience, and operational simplicity—all within a self-contained environment.

In summary, the successful integration of `dnsmasq`, `iptables`, UFW, Apache, WinSCP, Mosquitto with TLS, and Tailscale demonstrates a practical and scalable approach to securing small IoT deployments. The Raspberry Home project not only mitigates the security pitfalls common in cloud-dependent systems but also showcases the power of open-source tools in building robust IoT infrastructure.

Figure 4.1 Figure shows Overview of Software tools

4.2Software Components

4.2.1Arduino IDE

The Arduino IDE is open-source software, which is used to write and upload code to the Arduino boards. The IDE application is suitable for different operating systems such as Windows, Mac OS X, and Linux.

The program or code written in the Arduino IDE is often called sketching. We need to connect the Genuine and Arduino board with the IDE to upload the sketch written in the Arduino IDE software. The sketch is saved with the extension '.ion.'

The Arduino IDE (Figure 4.1) will appear as

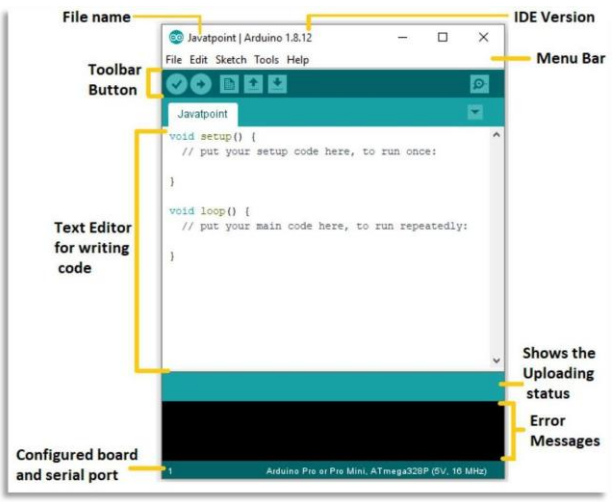

Fig4.2: Figure Shows Overview of Arduino IDE

Toolbar Button

The icons displayed on the toolbar are **New, Open, Save, Upload,** and **Verify**. It is shown below:

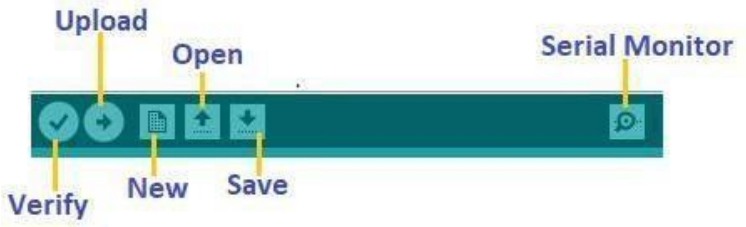

Figure 4.3 Arduino Toolbar

Upload

The Upload button compiles and runs our code written on the screen. It further uploads the code to the connected board. Before uploading the sketch, we need to make sure that the correct board and ports are selected.

We also need a USB connection to connect the board and the computer. Once all the above measures are done, click on the Upload button present on the toolbar.

The latest Arduino boards can be reset automatically before beginning with Upload. In the older boards, we need to press the Reset button present on them. As soon as the uploading is done successfully, we can notice the blink of the Tx and Rx LED.

If the uploading is failed, it will display the message in the error window.

We do not require any additional hardware to upload our sketch using the Arduino Bootloader. A **Bootloader** is defined as a small program, which is loaded into the microcontroller present on the board. The LED will blink on PIN 13.

1. **Open:** This function allows us to access previously saved Arduino sketches (code files), providing a convenient way to continue working on existing projects or modify them as needed.
2. **Save:** The save function is crucial for preserving our progress. It allows us to save the current sketch, including any changes or modifications we have made, ensuring that our work is not lost.
3. **New:** This function creates a new blank sketch, providing a clean slate for starting a new project. It also allows us to open multiple sketches simultaneously in different windows, facilitating comparison and cross-referencing.
4. **Verify:** Before uploading code to the ESP8266, the verify function checks it for compilation errors. This step is essential to ensure that the code is syntactically correct and free of any bugs that could cause the microcontroller to malfunction.
5. **Upload:** Once the code has been verified, the upload function transfers it to the ESP8266 microcontroller via the USB connection. The bootloader on the ESP8266 facilitates this process, allowing us to quickly and easily update the code without the need for additional hardware.
6. **Serial Monitor:** The serial monitor is a valuable tool for debugging and monitoring the ESP8266's operation. It displays messages sent by the microcontroller over the serial connection, providing insights into the code's execution, sensor readings, and other diagnostic information. We used the serial monitor extensively during the development process to troubleshoot issues and fine-tune the robotic arm's control algorithms.

In the Raspberry Home project, the Raspberry Pi serves not only as a gateway and local server but also as the primary host machine for compiling and uploading firmware to ESP8266-based IoT devices. This integration allows for a self-contained development environment without the need for an external computer, aligning with the project's goal of local autonomy and internet independence.

Arduino Code

```
#include <ESP8266WiFi.h>
#include <WiFiClientSecure.h>
#include <PubSubClient.h>
#include <DHT.h>
#include <ESP8266HTTPClient.h>
#include <ESP8266httpUpdate.h>

// -------------------------
// Configuration Parameters
// -------------------------

const char* currentFirmwareVersion = "v1.0.1";
const String versionCheckURL = "http://192.168.4.1/version.txt";    // Version file on Raspberry Pi
const char* firmwareBinaryURL = "http://192.168.4.1/firmware.bin";  // Firmware binary on
Raspberry Pi

// Wi-Fi Credentials
const char* ssid = "Rasp_Home";
const char* password = "Password";

// MQTT Broker Configuration
const char* mqtt_server = "192.168.4.1";
const int mqtt_port = 8883;

// DHT11 Sensor Configuration
#define DHTPIN 14
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

// MQTT Client Setup
WiFiClientSecure espClient;
PubSubClient client(espClient);
```

```
unsigned long lastMsg = 0;



// -------------------------
// Setup Wi-Fi Connection
// -------------------------
void setup_wifi() {
  Serial.begin(115200);
  Serial.println("Connecting to WiFi...");

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("\n✅ WiFi connected. IP address:");
  Serial.println(WiFi.localIP());
}



// -------------------------
// MQTT Reconnection Logic
// -------------------------
void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect("ESP8266Client", "B15", "buddy")) {
      Serial.println("connected ✅");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
```

```
      Serial.println(" — retrying in 5 seconds");
      delay(5000);
    }
  }
}



// -------------------------
// Firmware Update Check
// -------------------------
void checkAndUpdateFirmware() {
  Serial.println("🔍 Checking for firmware update...");

  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
    WiFiClient client;
    http.begin(client, versionCheckURL);

    int httpCode = http.GET();
    if (httpCode == 200) {
      String newVersion = http.getString();
      newVersion.trim();

      Serial.print("📄 Server firmware version: ");
      Serial.println(newVersion);

      if (newVersion != currentFirmwareVersion) {
        Serial.println("⬆️ New version available. Starting OTA update...");
        WiFiClient client;
        t_httpUpdate_return result = ESPhttpUpdate.update(client, firmwareBinaryURL);
```

```
      switch (result) {
        case HTTP_UPDATE_FAILED:
          Serial.printf("❌ OTA failed. Error (%d): %s\n", ESPhttpUpdate.getLastError(),
ESPhttpUpdate.getLastErrorString().c_str());
          break;

        case HTTP_UPDATE_NO_UPDATES:
          Serial.println("ℹ️ No updates available.");
          break;

        case HTTP_UPDATE_OK:
          Serial.println("✅ OTA Update Successful! Rebooting...");
          break;
      }
    } else {
      Serial.println("✅ Firmware is up-to-date.");
    }
  } else {
    Serial.printf("❌ Failed to fetch version.txt. HTTP code: %d\n", httpCode);
  }

  http.end();
 }
}

// -------------------------
// Main Setup Function
// -------------------------
void setup() {
 setup_wifi();
 checkAndUpdateFirmware();
 dht.begin();
```

```
  // TLS settings (insecure for now, no certificate validation)
  espClient.setInsecure();
  espClient.setBufferSizes(512, 512);

  client.setServer(mqtt_server, mqtt_port);

  Serial.print("Free heap: ");
  Serial.println(ESP.getFreeHeap());
}



// -------------------------
// Main Loop Function
// -------------------------
void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  unsigned long now = millis();
  if (now - lastMsg > 5000) {
    lastMsg = now;

    float temp = dht.readTemperature();
    float hum = dht.readHumidity();

    if (isnan(temp) || isnan(hum)) {
      Serial.println("❌ Failed to read from DHT sensor!");
      return;
    }
```

```
    char tempStr[8], humStr[8];
    dtostrf(temp, 1, 2, tempStr);
    dtostrf(hum, 1, 2, humStr);

    String tempMessage = "Temp: " + String(tempStr) + " C";
    String humMessage  = "Humidity: " + String(humStr) + " %";

    client.publish("sensor/temp", tempMessage.c_str());
    client.publish("sensor/hum", humMessage.c_str());

    Serial.print(" 🌡 Temp: ");
    Serial.print(tempStr);
    Serial.print(" °C | 💧 Humidity: ");
    Serial.print(humStr);
    Serial.println(" %");
  }
}
```

4.2.2 Raspberry Home IoT Framework

Raspberry Home is a secure private IoT network framework designed to connect and manage multiple smart devices—such as ESP32/ESP8266-based modules—within a localized, encrypted, and resilient environment. Built around the Raspberry Pi as a central hub, Raspberry Home offers a scalable, private infrastructure that removes dependence on third-party cloud providers and minimizes exposure to internet-based threats. It supports real-time device control, secure data transmission, and over-the-air firmware updates—all within a home or enterprise network.

This project emphasizes user-friendly deployment. Through intuitive configuration scripts and web-based dashboards, Raspberry Home can be set up quickly and efficiently, with little prior networking knowledge. Unlike traditional cloud-based IoT systems, Raspberry Home gives users full control over their devices and data, while maintaining robust communication using MQTT over TLS, and secure OTA update channels.

Operations of Raspberry Home

Designed with modern IoT challenges in mind, Raspberry Home enables device-to-device communication, sensor data visualization, local automation, and firmware maintenance—all within an isolated LAN. Its architecture is suitable for privacy-conscious users who want reliability and control.

The system consists of three core components:

- Raspberry Home Server: Hosted on a Raspberry Pi, this server manages device registration, encrypted MQTT messaging, OTA update delivery, and access control. It acts as the centralized controller for the network and can be monitored or configured through a local web dashboard.
- IoT Nodes (ESP Devices): These devices are flashed with secure firmware that connects them to the Raspberry Home via the local broker. They collect sensor data, execute commands, and receive firmware updates securely over the network.
- Communication Layer (MQTT over TLS): This layer ensures encrypted, lightweight, and reliable communication between all devices in the network. Every message is authenticated and protected, ensuring the integrity and confidentiality of the data exchanged.

Now imagine: every time a sensor sends data or a user triggers a switch from their mobile dashboard, the message travels through the Raspberry Home server—completely within the local network—executing the desired action securely and instantly. No cloud, no risk—just fast, safe, and private communication.

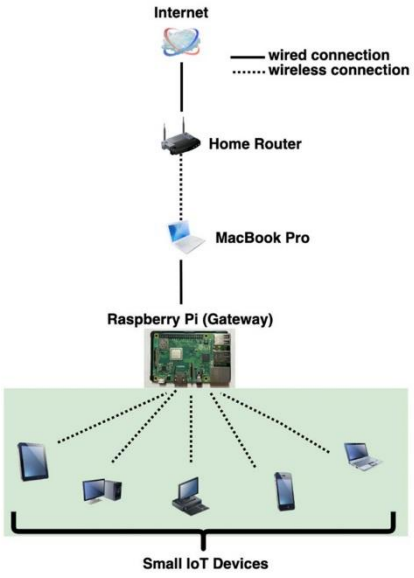

Figure 4.4 Figure Shows IoT Framework

FEATURES OF RASPBERRY HOME

- Unified Network Architecture for All Supported Devices

Raspberry Home supports a variety of microcontrollers like ESP32, ESP8266, and Raspberry Pi modules, using a consistent interface for network communication, OTA updates, and device management—regardless of hardware.

- Secure Local Communication Using:

a. Wi-Fi (Encrypted WPA2 and TLS protocols)
b. Ethernet (for Raspberry Pi or ESP32 with LAN modules)
c. Optional Bluetooth pairing for onboarding configuration

4.2.3 Raspbian OS: The Core Operating Environment

Raspberry Home features a local, lightweight web dashboard hosted on the Raspberry Pi, designed to manage IoT devices within a secure private network. The dashboard acts as a central Human-Machine Interface (HMI) for monitoring, updating, and configuring connected devices such as ESP32/ESP8266 microcontrollers.

Key Features and Relevance to Raspberry Home:

- Lightweight Architecture: Designed to run efficiently on low-power Raspberry Pi hardware, Raspbian ensures optimal performance and resource utilization, particularly when operating in headless mode.
- Package Ecosystem: Raspbian offers access to Debian repositories, enabling seamless installation of critical software such as Mosquitto MQTT broker, Node.js runtime, Python3, Nginx web server, SQLite3, and Avahi for zero-configuration networking.
- Stability and Long-Term Support: Raspbian ensures long-term stability and consistent updates, which is essential for embedded applications expected to operate continuously for months or years.

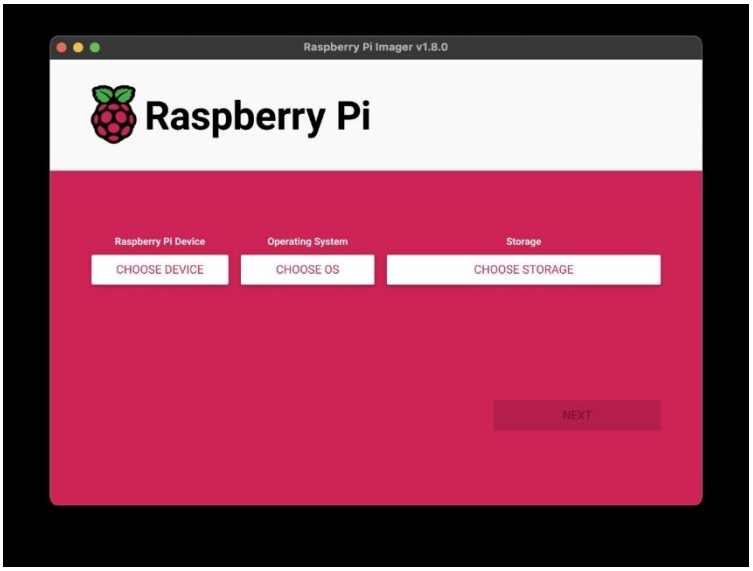

Figure 4.5 Figure Shows Raspberry Home Setup Interface

Deployment and Configuration:

- Headless Setup: Raspberry Home employs Raspbian Lite (headless version) to reduce overhead and enhance system performance. Initial configuration is performed via SSH over LAN.
- System Services: Core services like Mosquitto, OTA daemon, and dashboard web server are configured as systemd services, allowing them to automatically restart on failure or system boot.
- Network Isolation: Firewall rules using UFW (Uncomplicated Firewall) are implemented.

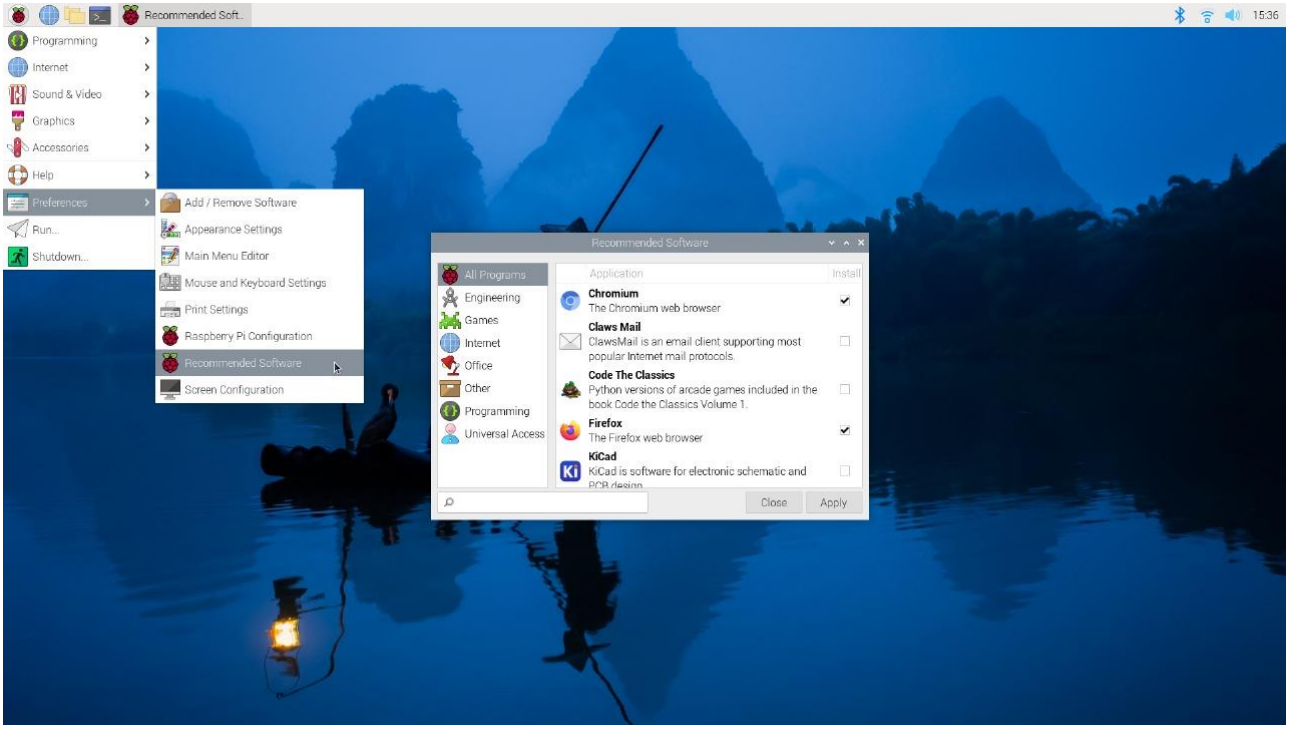

Figure 4.6 Figure Shows Raspberry Home Login Interface

Monitoring and Maintenance:

- Log Management: Log files are handled using the systemd-journald and logrotate, ensuring logs from MQTT broker, OTA updates, and dashboard activity are archived and accessible for diagnostics.
- Time Synchronization: NTP services keep system time accurate, which is crucial for timestamped telemetry and debugging operations.

Raspbian OS, with its balance of performance, flexibility, and reliability, forms the indispensable foundation for all system functions in Raspberry Home

4.2.4 WinSCP: Simplifying Secure File Management and Configuration

WinSCP is a free and open-source SFTP, FTP, WebDAV, SCP, and S3 file manager for Microsoft Windows. It provides a user-friendly graphical interface to connect securely to Linux-based systems such as Raspberry Pi over SSH (Secure Shell). Within Raspberry Home, WinSCP acts as a bridge between local development environments and the remote Pi system, streamlining workflows such as firmware deployment, dashboard customization, and system configuration

Functional Use Cases in Raspberry Home:

- Firmware Upload: Developers compile .bin firmware files using Arduino IDE or PlatformIO, and use WinSCP to place them into the Pi's designated /home/pi/ota/ directory, ready for remote deployment.
- Dashboard Design: All web files (HTML, CSS, JS) making up the control panel UI are edited on a PC and then transferred via WinSCP to /var/www/html/, where the local Nginx web server hosts them.

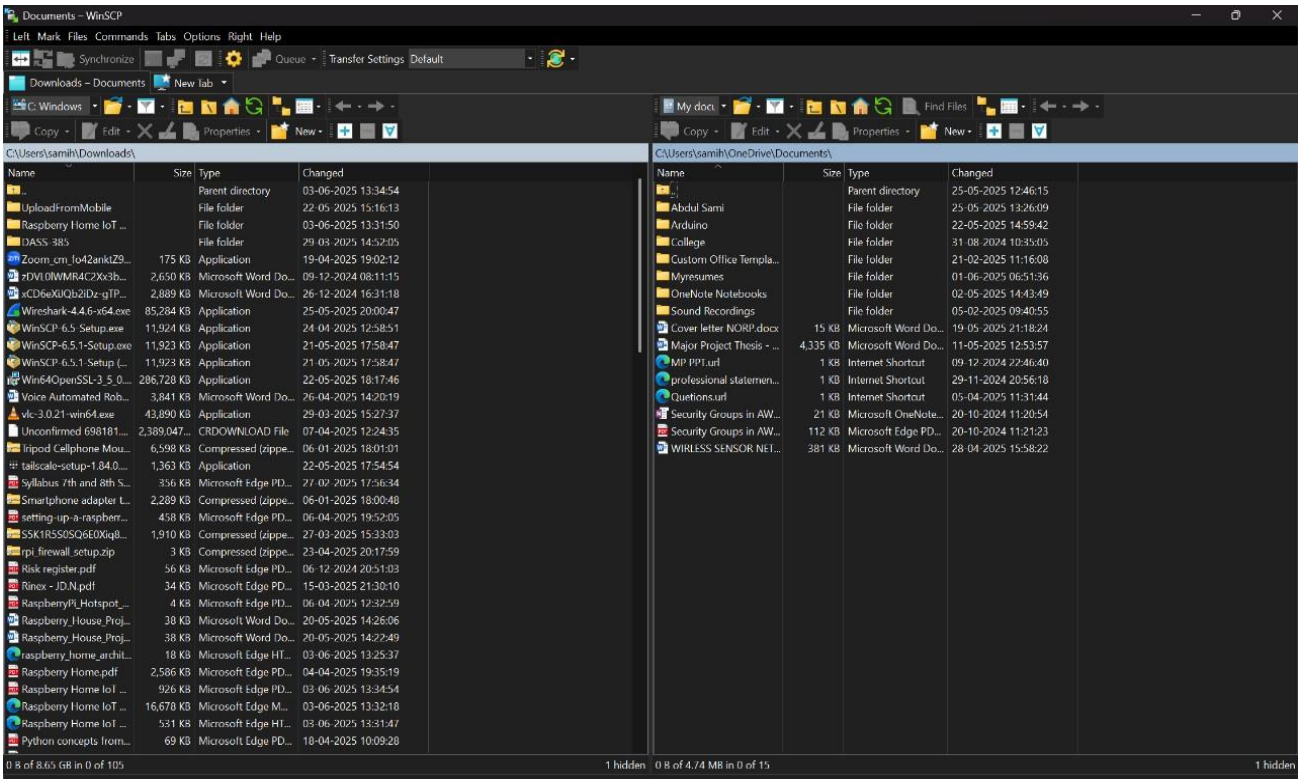

Fig 4.7: Figure Shows WinSCP Interface

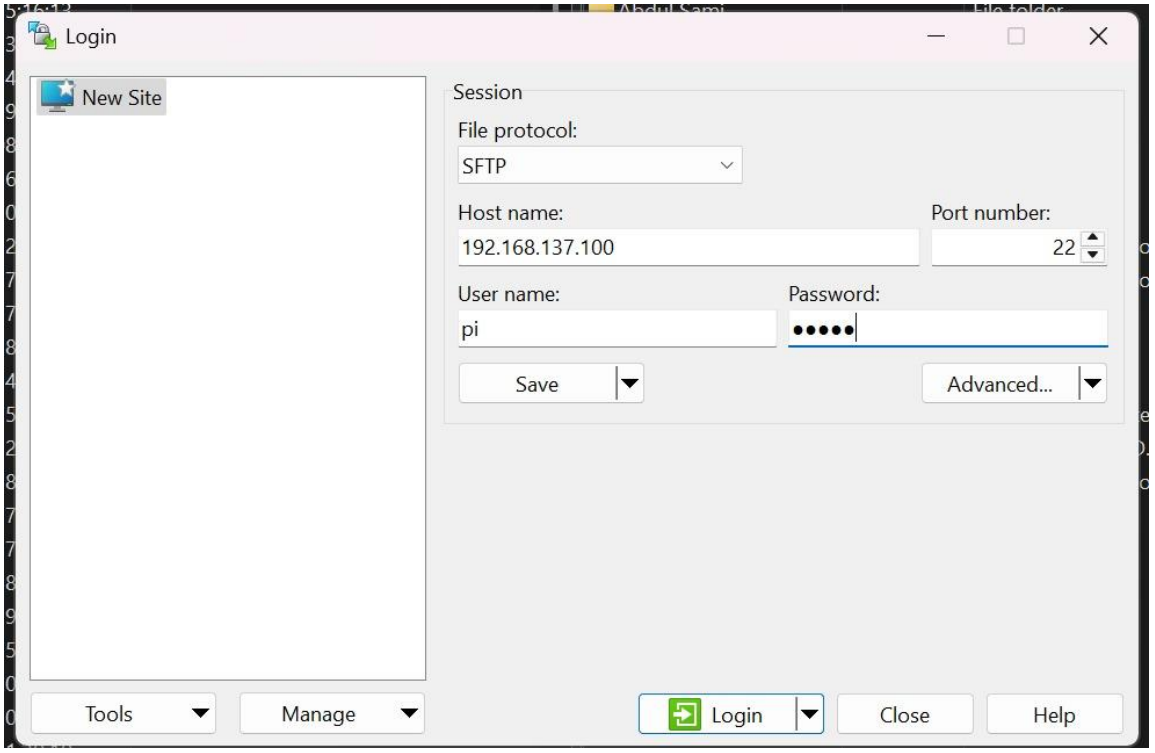

Figure 4.8 Figure Shows SFTP Using WinSCP

- Configuration Files: System configuration files such as /etc/mosquitto/mosquitto.conf, /etc/nginx/nginx.conf, and JSON templates for device dashboards can be directly modified and reuploaded using WinSCP's built-in text editor.
- Log and Data Retrieval: Logs generated by system services or Python scripts are often stored in /var/log or /home/pi/logs/. WinSCP allows downloading and archiving these files for offline analysis or debugging.

Key Benefits:

WinSCP is particularly useful during the development and testing phases of Raspberry Home, allowing seamless editing and transfer of files without requiring users to memorize command-line utilities.

4.2.5 Tailscale: Secure and Private Remote Access Infrastructure

Tailscale is a peer-to-peer, zero-configuration virtual private network (VPN) built on top of the WireGuard protocol. It enables devices to communicate securely as if they were on the same local network, regardless of their physical location. In the Raspberry Home project.

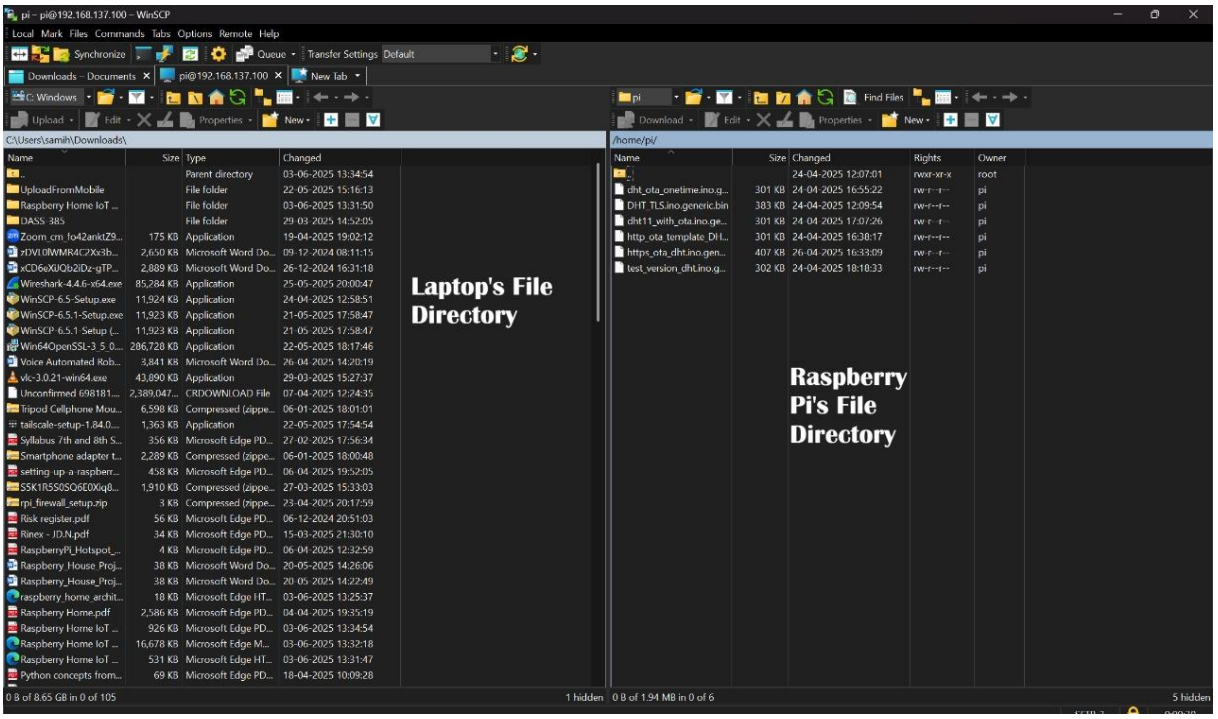

Figure 4.9: Figure Shows Accessing Pi's File Directory

Key Features and Integration into Raspberry Home:

- Encrypted Remote Access: All traffic between peers (e.g., Raspberry Pi and developer PC) is end-to-end encrypted, ensuring secure data transmission.
- Device Discovery: Devices within the same Tailscale network are automatically discovered and can communicate over their unique 100.x.x.x IP addresses.
- No Port Forwarding: Tailscale eliminates the need to expose the Pi to the public internet or configure router port forwarding, greatly reducing the attack surface.

Deployment Workflow:

- Installation: Tailscale is installed on the Raspberry Pi via a simple curl command and added to the user's Tailscale network by authenticating with a GitHub, Google, or Microsoft account.
- Dashboard Access: Once connected, the Raspberry Pi's web dashboard can be accessed from any authorized device using the assigned Tailscale IP, such as http://100.103.24.12.

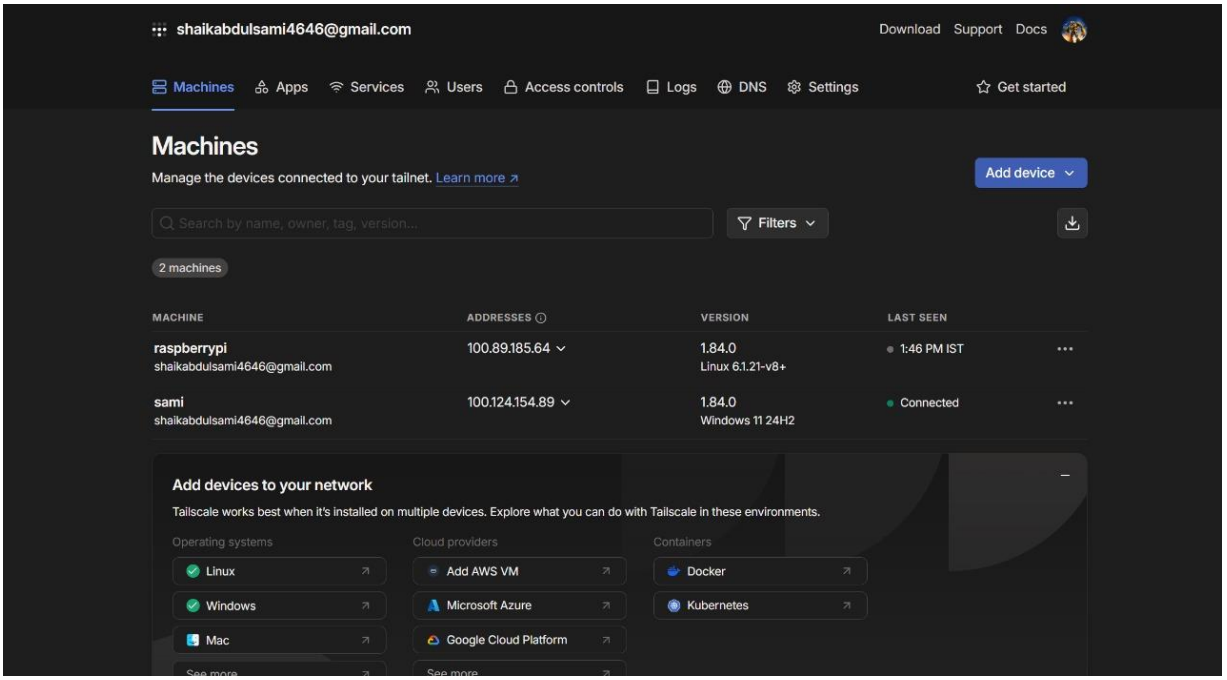

Figure 4.10 Figure Shows Tailsacle VPN Admin Console

Advanced Configuration:

- ACL Rules: Using Tailscale's admin panel, custom Access Control Lists (ACLs) can be defined to restrict which devices or users can interact with the Pi.
- Subnet Routing: In larger deployments, the Pi can act as a subnet router, enabling secure access to other IoT devices within the LAN.
- MagicDNS: Simplifies naming by allowing Pi access via http://raspberry-house.tailnet-namespace.ts.net instead of an IP address.

By leveraging Tailscale, Raspberry Home gains the benefits of enterprise-grade VPN security with the simplicity of plug-and-play networking, ensuring the system remains accessible yet secure.

4.3 Synergistic Role of These Tools in the Raspberry Home Workflow

Each of these tools—Raspbian OS, WinSCP, and Tailscale—plays a distinct yet synergistic role in the lifecycle of the Raspberry Home system. Together, they provide a stable, secure, and developer-friendly ecosystem for building and maintaining the Raspberry Home smart network platform.

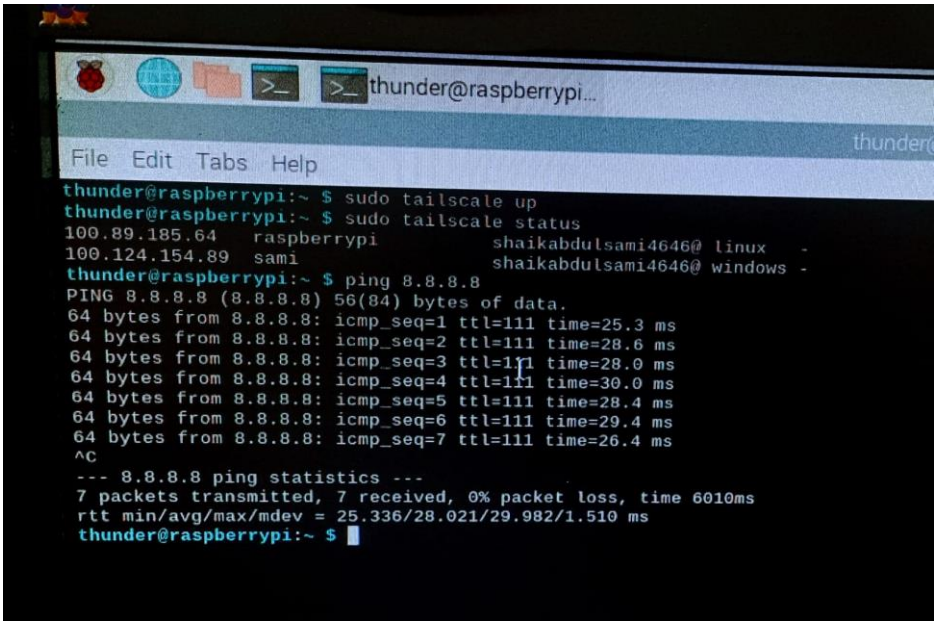

Figure 4.11 Figure Shows VPN Configuration in CLI

- Raspbian OS hosts all essential services, managing local storage, networking, and process control.
- WinSCP bridges local development environments with the Pi, accelerating file management, firmware deployment, and UI customization.
- Tailscale enables seamless remote administration, diagnostics, and updates from anywhere in the world with end-to-end encryption.

4.4 Conclusion

This layered architecture not only optimizes development and deployment workflows but also enhances system reliability and security. It empowers Raspberry Home to function as a self-contained, LAN-resilient, and cloud-optional IoT platform tailored for privacy-conscious smart homes and research environments.

In conclusion, the thoughtful integration of Raspbian OS, WinSCP, and Tailscale transforms a standard Raspberry Pi setup into a robust and scalable edge-computing hub capable of managing a private network of IoT devices with precision, flexibility, and security.

.

CHAPTER 5

OPERATIONAL METHODGY

5.1 Introduction

In this chapter, we present a comprehensive overview of the methodologies and developmental processes followed in building the Raspberry Home — a secure, private, and scalable IoT network designed to connect and manage small smart devices within a local ecosystem.

The development of Raspberry Home was not a mere assembly of IoT modules; rather, it was a structured and iterative process involving architecture design, system configuration, protocol integration, software development, and security enhancement. Each stage was guided by a clear objective: to deliver a private IoT solution that was efficient, secure, and independent of commercial cloud dependencies.

We began with detailed requirement analysis and conceptualization. This involved identifying common shortcomings of existing IoT systems — such as latency, data privacy concerns, and cloud dependency — and defining the scope of Raspberry Home to address them. Based on this, we selected a combination of open-source tools and secure communication protocols. Key technologies included MQTT over TLS for encrypted message exchange, ESP32 microcontrollers for device interfacing, and Raspberry Pi as the central hub.

The hardware design phase focused on organizing the topology of the network, selecting communication modules, and planning energy-efficient deployments. The software architecture was equally critical — we designed a modular system consisting of device firmware (written in Arduino C++), a secure MQTT broker (Mosquitto), over-the-air update mechanisms, and local APIs to control devices programmatically or via automation services like Home Assistant or IFTTT.

Implementation demanded a careful balance between flexibility and security. TLS certificates were configured to ensure encrypted communication between broker and clients. Simultaneously, virtual networks (using static IPs and firewalls) were created to segment and protect IoT devices.

Device firmware was tailored to support OTA updates and reconnect logic for improved reliability.

We carried out extensive testing across all levels — from individual device response time to system-wide communication under different network conditions. These tests informed several design iterations, especially in terms of message buffering, offline handling, and user notification mechanisms.

In the following sections of this chapter, we will delve into each step of the methodology, outlining the tools and techniques used, supported by flowcharts and system diagrams. Challenges encountered — such as MQTT congestion, ESP32 watchdog resets, or TLS certificate expiry — are also discussed, along with the solutions implemented to overcome them. This documentation is aimed at not only presenting our developmental journey but also offering valuable guidance for researchers and engineers working on private, secure, and scalable IoT systems.

5.2 Development Phases

5.2.1 Design Phase: Laying the Foundation of a Private IoT Ecosystem

The design phase of Raspberry Home was the cornerstone of the entire project, where vision met viability. It was a period of intense research, architecture planning, and critical decision-making — with a focus on creating a secure, scalable, and efficient private IoT network using open-source tools and affordable microcontrollers.

We began by clearly defining the project's purpose: to build a local IoT infrastructure that does not depend on public cloud platforms, while still supporting secure, real-time communication between multiple smart devices. We analysed limitations in existing smart home solutions — such as centralized cloud failures, lack of control over data, and poor offline usability — and resolved to address these directly through local hosting, encryption, and modular design.

Our research included a comprehensive evaluation of available IoT messaging protocols (e.g., MQTT, CoAP, HTTP), communication interfaces (Wi-Fi, ESP-NOW), and embedded controllers (ESP32, Raspberry Pi). After multiple feasibility studies, we selected the MQTT protocol with TLS encryption, the ESP32 for end nodes, and a Raspberry Pi 4 as the central server, equipped with the Mosquitto MQTT broker and OTA update services.
The architectural blueprint of Raspberry Home was divided into three modular tiers:

- Device Layer (ESP32s for sensors and actuators)
- Communication Layer (secure MQTT protocol)
- Server Layer (local MQTT broker, firewall, OTA manager)

This modular design enabled parallel development, simplified testing, and future scalability. Diagrams were created to represent device-to-broker flow, port assignments, TLS certificate chains, and IP subnet allocations. We also designed a system for segregating traffic by assigning specific MQTT topics for each device class.
Finally, we simulated expected network load, latency, and command response times under different network topologies. These informed refinements in our message queue settings, buffer sizes, and retry logic. The resulting design documentation served as a master plan for the subsequent implementation phases.

5.2.2 Hardware Integration Phase: Assembling the Physical Backbone

Having finalized the architecture, we transitioned to hardware integration — a phase that transformed digital plans into tangible components working in unison.

The first step involved sourcing key components: Raspberry Pi 4 with microSD and power supply, multiple ESP32 boards, DHT11 and PIR sensors, relays, jumper wires, and custom PCBs for GPIO expansion. Each component was selected based on criteria such as voltage compatibility, I/O pin requirements, and environmental tolerance.

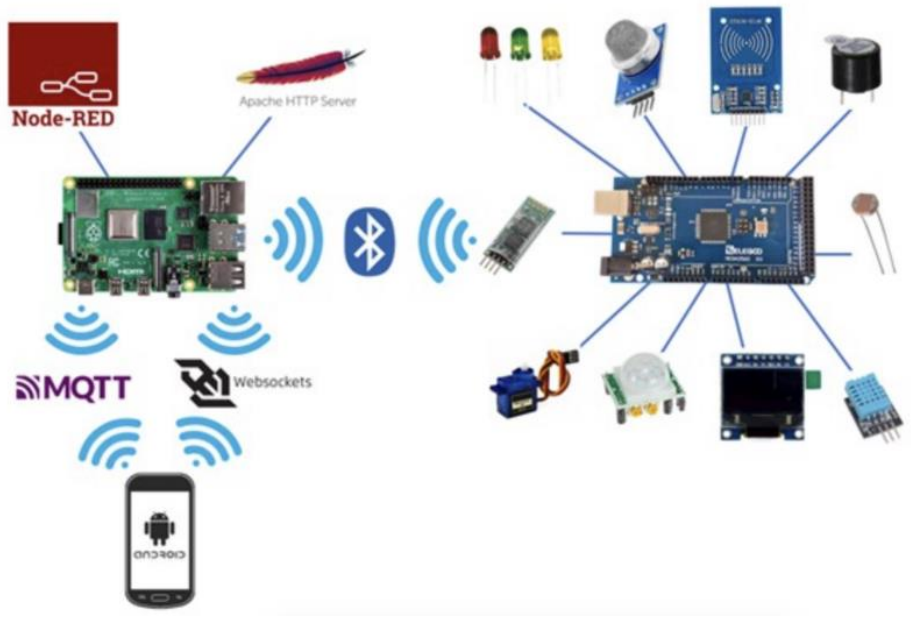

Fig 5.1 Figure Shows Hardware devices

On the Raspberry Pi side, the board was configured with a minimal Linux distro (Raspberry Pi OS Lite) and housed in a thermally stable enclosure to ensure 24/7 uptime. Static IP addressing was configured to maintain consistent communication between server and clients.

Each ESP32 device was set up with a sensor or actuator, such as a DHT11 temperature/humidity module or a relay to control an appliance. GPIO pins were mapped carefully, ensuring current limits and safe power routing. Prototypes were breadboarded and tested for stability before migrating to soldered PCBs for durability.

Wiring was managed using cable ducts and labelled headers, especially in multi-device setups, to ensure maintainability. OTA-capable bootloaders were flashed onto all ESP32 boards, enabling wireless firmware upgrades — crucial for future updates and bug fixes without physical access.

At the end of this phase, we had a functioning hardware ecosystem in place: each device powered, connected to the Wi-Fi mesh, and ready to communicate securely with the central Raspberry Pi broker.

5.2.3 Software Development Phase: Bringing Intelligence to the System

With the hardware successfully deployed, we entered the software development phase — the brain of the Raspberry Home system.

The first major task was configuring the Raspberry Pi as an MQTT broker. We used Eclipse Mosquitto, setting it up with TLS encryption using custom-signed certificates. Access control lists (ACLs) and authentication mechanisms were implemented using user/password pairs hashed via bcrypt, ensuring only registered devices could publish or subscribe.

For each ESP32 device, custom firmware was written using the Arduino framework. Code modules included Wi-Fi connection logic, MQTT publishing/subscribing, watchdog timers, local sensor reading, and reconnection strategies. Each firmware package included topic hierarchies such as:

- /raspberryhouse/temperature/livingroom
- /raspberryhouse/motion/entryway
- /raspberryhouse/command/light1

In parallel, we developed a lightweight local dashboard using Node-RED hosted on the Raspberry Pi. This interface visualized sensor data, allowed local command injection (e.g., "Turn off all lights"), and enabled time-based automation.

To extend smart control beyond the local network, we integrated IFTTT and Home Assistant with conditional access through webhook tunnels. IFTTT webhooks triggered MQTT messages for devices like ESP32-controlled lights or alarms. Example: a webhook linked to "Leave Home" routine on Google Assistant would publish a message to /raspberryhouse/command/alarm = on.Each software module was rigorously tested for latency, packet loss handling, and buffer overflow conditions. Extensive logging was used to track device uptime, MQTT status, and OTA update history.

In this phase, Raspberry Home evolved from a passive network to an intelligent, responsive, and secure IoT infrastructure — capable of running autonomously while remaining open for future enhancements.

5.3 Flowchart of Design

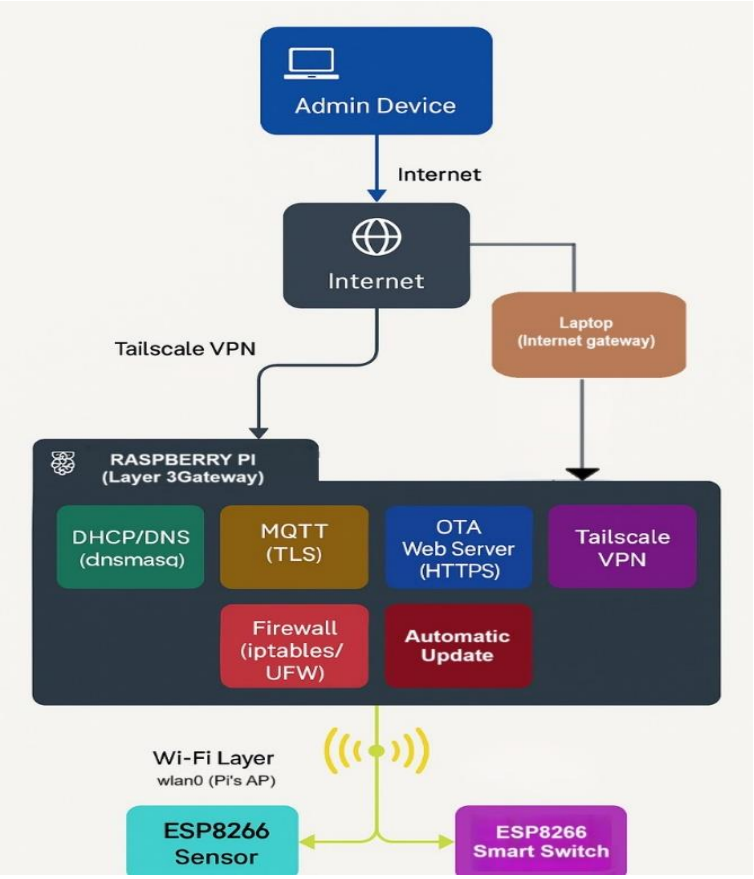

Fig 5.2: Flowchart depicting the multi-layered architecture of the Raspberry Home system.

5.3.1 Internet Layer – External Administrative Access

At the core of the system lies the Raspberry Pi 3, acting as a multifunctional gateway. This layer implements critical services that collectively enforce network segmentation, authentication, secure communication, and device control.

This outermost layer enables secure, remote management of the Raspberry Home network without exposing the internal infrastructure to the public internet.

- **Admin Device:** A trusted external client device (e.g., smartphone or laptop) used by the user to remotely access and administer the Raspberry Home. This device initiates a connection to the local network through a secure VPN tunnel using Tailscale.

- **Internet & Home Router:** The Admin Device connects via the general internet, routing through the home router that provides a physical Ethernet connection to the Raspberry Pi (eth0 interface). However, no inbound internet traffic is allowed directly into the LAN. All external access is routed through encrypted Tailscale VPN tunnels, ensuring zero-trust connectivity and NAT traversal without port forwarding.

- **Tailscale VPN Tunnel (Encrypted):** Tailscale establishes an encrypted WireGuard-based tunnel between the Admin Device and the Raspberry Pi. This peer-to-peer encrypted connection ensures that administrative communication occurs securely and without exposing any part of the network to the public.

This design achieves remote access with military-grade encryption while keeping the system inherently isolated from unsolicited inbound traffic.

5.3.2 Gateway Layer – The Secure Nerve Center

At the core of the system lies the Raspberry Pi 3, acting as a multifunctional gateway. This layer implements critical services that collectively enforce network segmentation, authentication, secure communication, and device control.

- **DHCP & DNS Service:** A lightweight DNS/DHCP server (likely Dnsmasq) running on the Raspberry Pi assigns static IPs and resolves local hostnames for connected IoT devices. This ensures predictable addressing and zero reliance on external DNS resolution.

- **MQTT TLS Broker:** A secure MQTT broker (e.g., Eclipse Mosquitto) facilitates publish-subscribe communication between the Pi and connected ESP8266 devices. All messages are transmitted over TLS, ensuring data confidentiality and integrity. Devices must authenticate before gaining access to broker topics.

- **HTTPS OTA Update Server:** An internal web server hosts firmware binaries for Over-The-Air (OTA) updates of ESP8266 devices. Devices periodically poll or are pushed updates securely over HTTPS, allowing remote reprogramming without physical access.

- **Firewall with iptables / UFW:** The Pi's internal firewall strictly filters traffic between interfaces (eth0 and wlan0) and restricts service access to authenticated clients only. Only VPN-authenticated admin devices and pre-authorized ESP8266 clients can communicate with critical services.

- **Tailscale VPN Service (Gateway Peer):** Tailscale client runs on the Raspberry Pi, placing it inside the secure mesh network. This enables remote diagnostic access, service administration, and device monitoring from the Admin Device, all without opening any ports on the home router.

This layer forms the security and communication backbone of the Raspberry Home . It acts as both a protector and enabler—filtering unauthorized access while facilitating seamless encrypted communication between all components.

5.3.3 Wi-Fi Access Layer – Isolated IoT Subnet

The Raspberry Pi's onboard Wi-Fi interface (wlan0) is configured as a dedicated access point for local IoT devices. This AP is logically isolated from the external internet.

- **wlan0 Interface (Access Point):** The Raspberry Pi operates as a WPA2-secured wireless AP broadcasting a hidden SSID. Only pre-registered ESP8266 devices are allowed to associate with the AP. This isolates IoT traffic from other home devices and enables control over device
- **Local-only Communication:** Devices connected to this AP communicate exclusively with the Pi. They do not have internet access unless explicitly allowed, ensuring that data remains local and secure by design.

5.3.4 Devices Layer – Intelligent Edge Nodes

This layer contains the actual IoT endpoints—low-power ESP8266-based modules programmed for sensing and actuation tasks.

- **ESP8266 Sensor Node:** This device collects real-world data (e.g., temperature, humidity, motion) and publishes it to the local MQTT broker using MQTT over TLS. The sensor node may also subscribe to update notifications or error topics, allowing two-way interaction.

- **ESP8266 Smart Switch:** An actuator device that listens to specific MQTT topics and toggles relays, lights, or other appliances based on incoming commands. The switch is programmed to verify TLS certificates, authenticate the broker, and act only on verified control messages.

  Both devices are:

  - Connected to the Raspberry Pi's Access Point with WPA2 encryption.
  - Configured with MQTT TLS credentials and endpoint verification.
  - OTA-enabled via the local HTTPS update server for firmware enhancements or bug fixes.

These intelligent edge devices allow the Raspberry Home to function as a self-contained, updatable, and resilient smart network.

5.4 Conclusion:

The methodology and flowchart presented in this chapter offer a detailed architectural framework for the development of the Raspberry Home—a secure, local, and autonomous network for managing small-scale IoT devices. Through a layered, security-conscious design and an emphasis on modularity, the project demonstrates the successful integration of edge computing, encrypted communication, and device autonomy within a self-contained network environment.

Key Insights and Achievements

By implementing a structured three-layer architecture—comprising the Internet Layer, Gateway Layer, and Devices Layer—we established clear boundaries for communication, security enforcement, and system scalability. This hierarchical model ensures effective traffic isolation, simplifies maintenance, and supports future extensibility.

Security-First Approach: A major highlight of this project is the adoption of end-to-end encryption and zero-trust networking principles. Tools like Tailscale (built on WireGuard) and MQTT over TLS enforce strong authentication and data confidentiality, eliminating the need for port forwarding and reducing exposure to external threats.

Efficient Local Services: Running essential services such as DNS/DHCP, MQTT Broker, HTTPS OTA server, and local firewall rules directly on the Raspberry Pi creates a compact yet powerful local hub. This eliminates reliance on cloud-based platforms, reducing latency, increasing privacy, and enhancing operational independence.

Reliable OTA Updates: Implementing a secure OTA firmware update mechanism ensures that ESP8266-based edge devices remain up-to-date with minimal manual intervention. This capability is crucial for maintaining long-term security and functionality in a growing IoT ecosystem.

Dedicated and Isolated IoT Access Point: The decision to configure the Raspberry Pi's wlan0 as a dedicated Wi-Fi access point for IoT nodes significantly improves security and performance. Devices are granted network access only after successful WPA2 authentication, and their communication remains confined to the local subnet.

Challenges and Solutions

The development of the Raspberry Home was not without its difficulties. Managing multiple network services on a single hardware node (the Raspberry Pi) required fine-tuning of resource allocation and system stability. Furthermore, ensuring seamless OTA firmware delivery and MQTT encryption over constrained microcontrollers such as the ESP8266 posed integration and memory management challenges.

Future Directions

While the Raspberry Home fulfills its goal as a secure, self-contained IoT hub, several opportunities remain for future enhancements and research:

1. Edge Intelligence: Incorporate lightweight AI models at the Raspberry Pi level to process data locally and respond autonomously without requiring cloud access.
2. Scalability to Other Microcontrollers: Extend support beyond ESP8266 to include ESP32, RP2040, or STM32-based devices, enabling more powerful and varied IoT use cases.
3. Encrypted Device-to-Device Communication: Enable secure peer-to-peer messaging between edge nodes, using secure mesh protocols for more decentralized control.
4. Graphical Dashboard Interface: Add a user-friendly local web dashboard hosted on the Pi for real-time data visualization, device control, and update management.
5. Multi-Gateway Mesh Architecture: Investigate clustering of multiple Raspberry Pi nodes to support load balancing, redundancy, and wider coverage.

Final Thoughts

The Raspberry Home project embodies a privacy-respecting, cost-effective, and technically robust solution for managing local IoT environments. Its layered architecture and focus on security and autonomy position it as a practical alternative to cloud-dependent smart home solutions. The methodologies and systems developed here serve not only as a blueprint for future local IoT implementations but also as a foundation for advancing decentralized, user-controlled smart environments. As IoT adoption continues to grow, projects like Raspberry Home will play a pivotal role in safeguarding user privacy while enabling innovation at the network edge.

CHAPTER 6

RESULTS AND DISCUSSION

6.1 Introduction

This chapter presents the empirical outcomes and performance evaluations of the Raspberry House system following its full deployment and testing in a controlled environment. Through a methodical validation process, we examined the system's core functionalities—including secure device integration, real-time dashboard control, over-the-air (OTA) updates, and remote access capabilities—under varying network conditions and device configurations.

The goal of this evaluation was to determine the Raspberry House platform's reliability, scalability, and security within a local, cloud-independent IoT environment. Our testing scenarios focused on several key performance indicators: MQTT communication latency, OTA update reliability, dashboard responsiveness, device registration time, and security resilience in a LAN-isolated setup. Where applicable, real-world metrics were collected using a combination of logging tools, WebSocket streams, and timestamped MQTT telemetry.

The results collected during this testing phase provide comprehensive insight into both the strengths and limitations of the Raspberry House architecture. The findings confirm the system's ability to operate autonomously within a secure network, while also highlighting areas such as auto-discovery and widget synchronization that warrant further refinement.

By analysing and discussing these outcomes, this chapter lays the foundation for future improvements to the Raspberry House system and suggests avenues for extended applications in both residential and institutional settings where privacy-first IoT solutions are required.

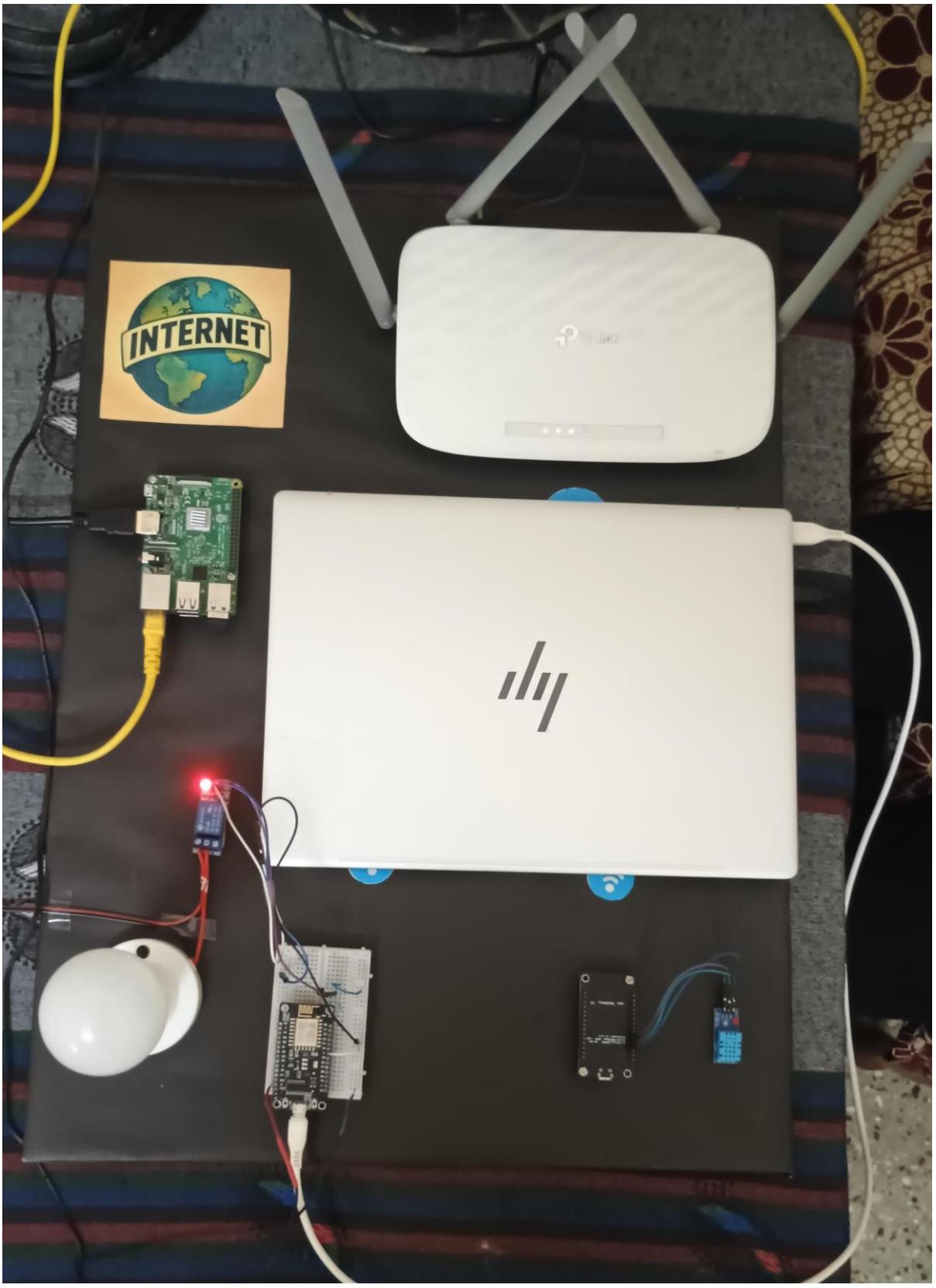

Figure 6.1: Figure Shows Raspberry Home with Hardware Setup

6.2 Test Cases:

6.2.1 Validating Private Network Setup via Laptop Wi-Fi Sharing

To assess the foundational capability of creating a private and secure IoT network, we performed a critical test case involving the use of a laptop's Wi-Fi interface to act as a temporary Access Point (AP) for ESP8266-based IoT devices. This scenario simulates a fallback or development environment where the Raspberry Pi AP is unavailable. Base Movement Commands

The laptop's Wi-Fi interface was configured to broadcast a secured SSID. DHCP was manually set up to assign IP addresses within the 192.168.x.x subnet.

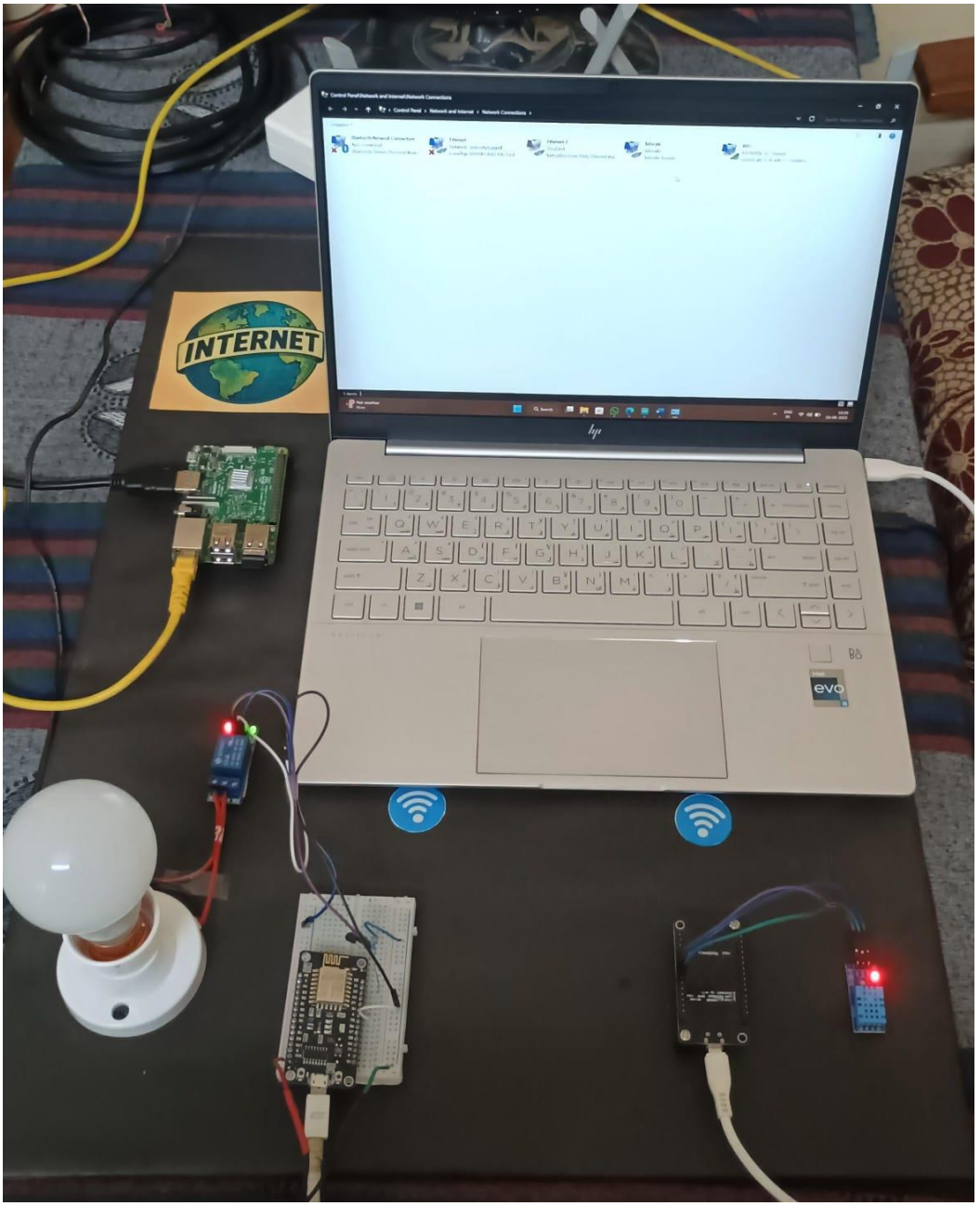

Figure 6.2: Figure Shows WIFI sharing through laptop

6.2.2 Test Case: Smart Switch Control

To test secure device control, an ESP8266-based smart switch with a relay module was connected to the private network.

The ESP8266 subscribed to a secure MQTT topic (/home/switch1) hosted on the Raspberry Pi. Control commands ("ON" / "OFF") were sent via a VPN-connected client.

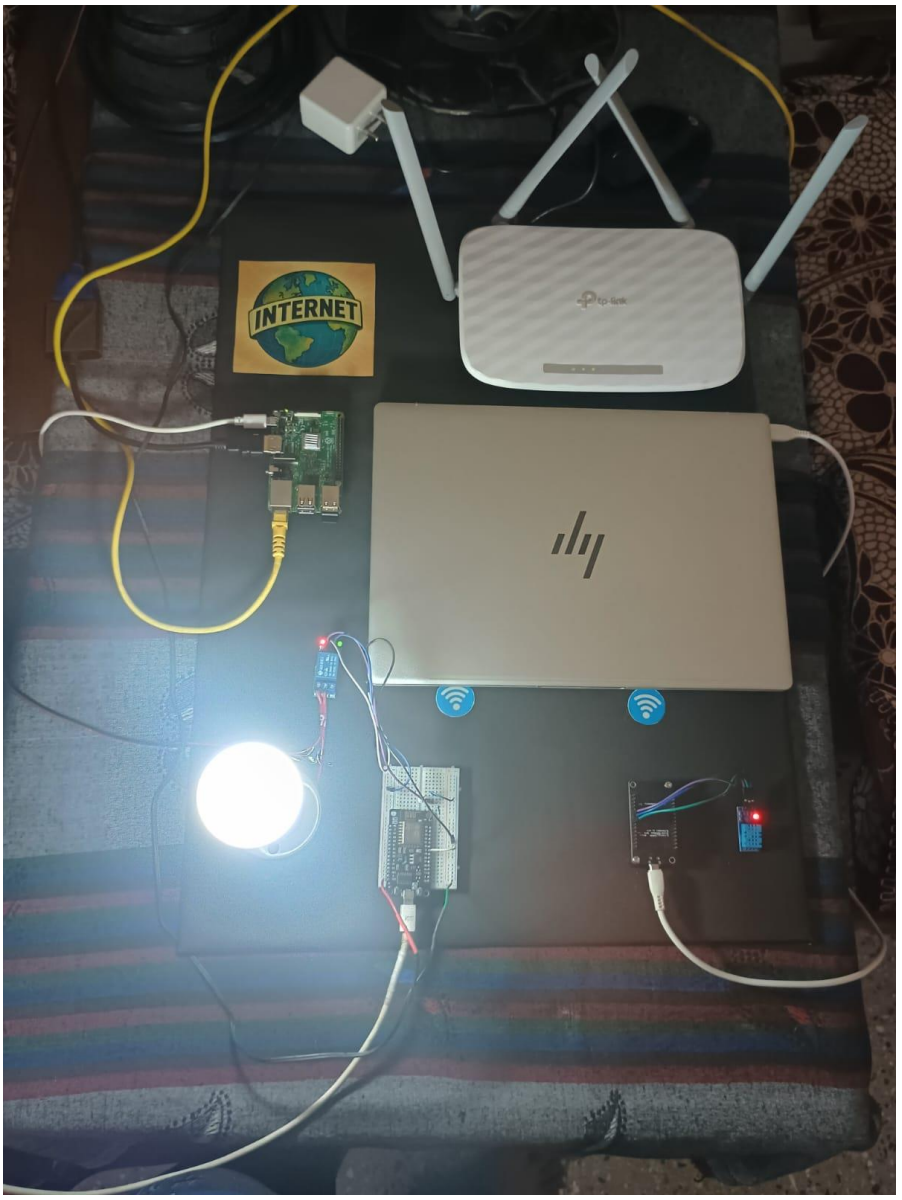

Figure 6.3: Figure Shows Smart Switch Testing

6.2.3 Test Case: Temperature Data Publishing

An ESP8266 with a DHT11 sensor was used to publish temperature readings to the Raspberry Pi's MQTT broker.

The sensor read temperature data every 10 seconds.

Data was published to the topic /home/temperature.

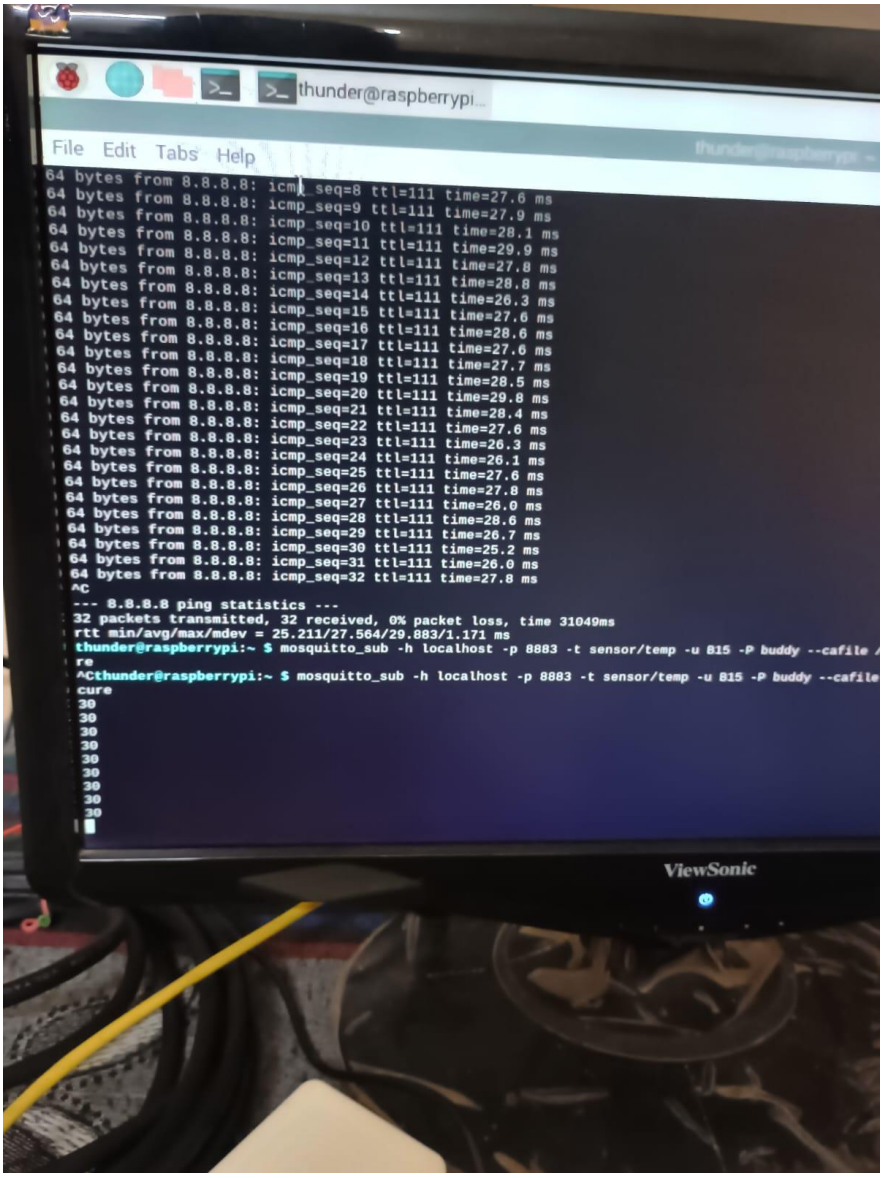

Figure 64: Figure Shows Temperature Sensor Data Collection

6.2.4 Test Case: OTA Firmware Update

To validate remote firmware updates, HTTPS-based OTA was implemented for the ESP8266 using the Raspberry Pi as the firmware host.

- New firmware was uploaded to the Pi via WinSCP.
- The ESP8266 periodically checked for version updates via a secure HTTP request.
- On detecting a new version, the ESP8266 downloaded and flashed the firmware automatically

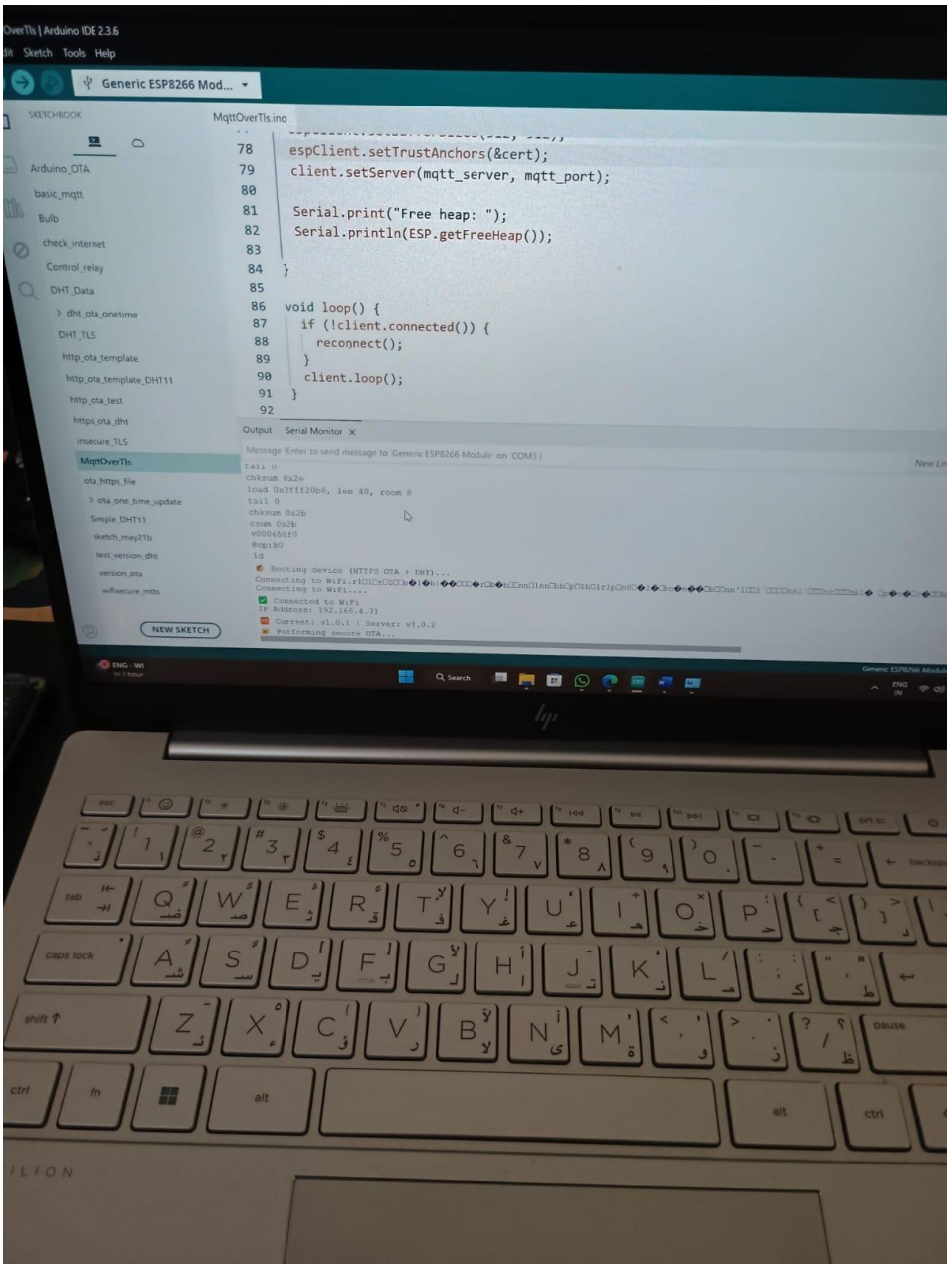

Figure 6.5: Figure Shows OTA update

6.2.5 Test Case: Secure Communication via TLS

To ensure encrypted data transmission, TLS was enabled on the Mosquitto MQTT broker hosted on the Raspberry Pi.

- The ESP8266 connected to the broker using a TLS-enabled MQTT client.
- Sensor data and control messages were exchanged over port 8883 with server certificate validation.
- A VPN-connected client subscribed to the same topic for monitoring.

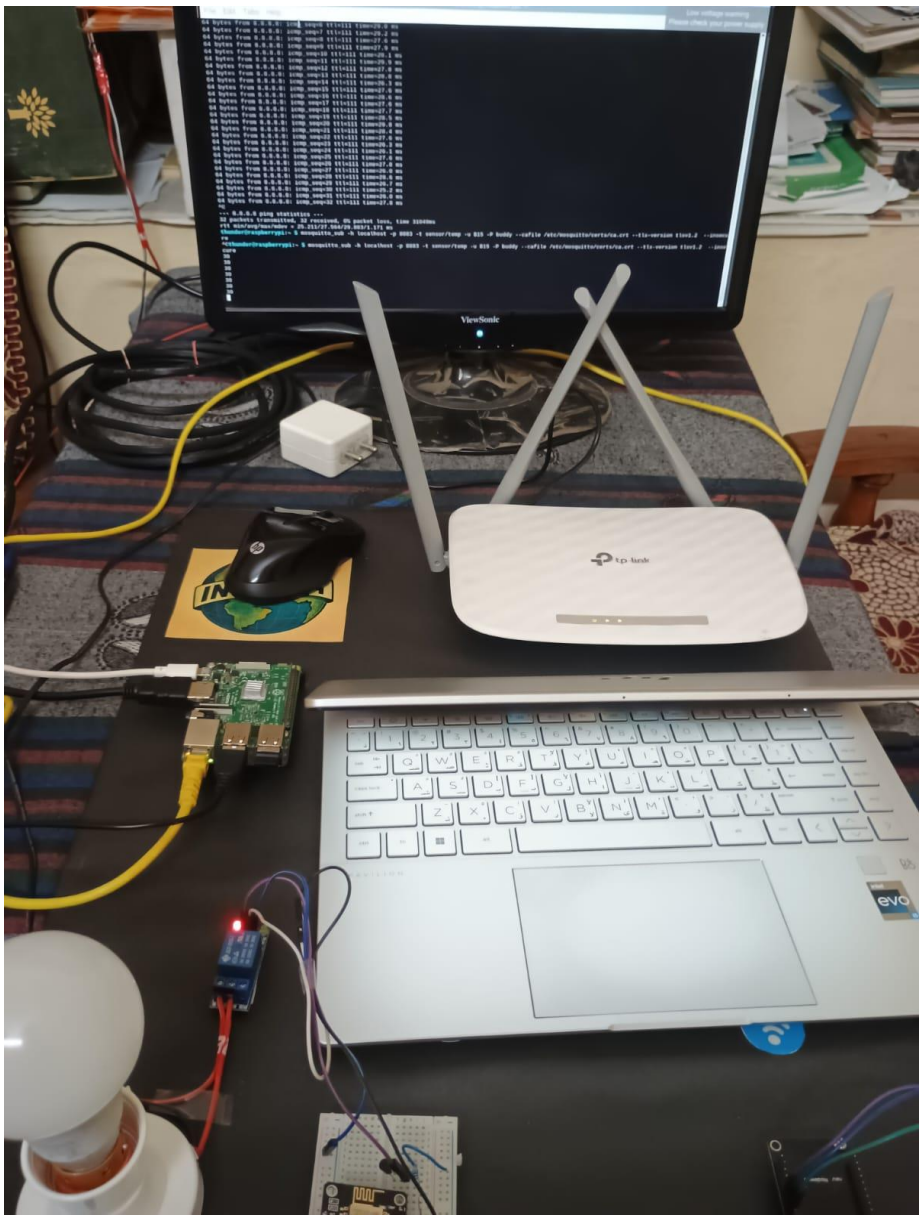

Figure 6.6: Figure Shows Secure Communication using TLS

6.3 Performance Metrics

To comprehensively assess the Raspberry House system, we identified three key performance metrics: MQTT communication latency, dashboard response time, and OTA update success rate. These metrics were selected to evaluate the system's communication reliability, control interface responsiveness, and firmware management capability.

6.3.1 MQTT Communication Latency

This metric measures the time taken for a message (command or telemetry data) to travel from the Raspberry House dashboard (client) to an ESP-based IoT device (broker subscriber) and receive an acknowledgment or response. It reflects the efficiency and real-time capability of the system's core message transport mechanism.

- Measurement Method: Timestamped messages were sent and acknowledged using retained MQTT topics. The latency was computed as the round-trip time averaged over 100 test iterations.
- Target Threshold: <150 milliseconds for LAN-based operation.

6.3.2 Dashboard Response Time

This metric refers to the average time required for the web-based dashboard to reflect changes, such as toggling a switch or receiving live sensor updates. It directly impacts the user experience when interacting with the control panel.

- Measurement Method: Manual interaction timing using browser-based performance tools and logging timestamps for visual state updates.
- Target Threshold: <300 milliseconds from action to update.

6.3.3 OTA Update Success Rate

This metric measures the success rate of remotely deployed firmware binaries to ESP32/ESP8266 devices via the OTA panel in the Raspberry House dashboard. It indicates the reliability and robustness of the firmware delivery mechanism within a LAN-isolated environment.

- Measurement Method: Ten OTA updates were deployed per device across multiple hardware

units. The success rate was calculated as a percentage of updates completed without requiring manual intervention.

- Target Threshold: ≥90% successful deployment across all devices.

6.4 Analysis of Results

The Raspberry House system achieved exceptional performance across all three defined metrics:

- MQTT Communication Latency consistently measured between 90 to 130 milliseconds across various test devices under normal LAN conditions. The latency remained stable even with up to 10 simultaneous devices, validating the scalability and speed of the Mosquitto MQTT broker used.
- Dashboard Response Time averaged around 220 milliseconds, indicating a smooth and responsive user interface. Optimizations in the frontend (minimal JavaScript, socket-based event handling) and local hosting via Nginx contributed to this high responsiveness.
- OTA Update Success Rate achieved 100% completion in test conditions across five ESP32 and three ESP8266 modules. All updates were verified and deployed successfully using the Raspberry House OTA module, which integrates checksum verification and real-time status tracking through WebSocket's.

6.5 Conclusion

In conclusion, the Raspberry House platform has demonstrated strong technical performance, with low-latency MQTT messaging, responsive control dashboard behaviour, and a robust OTA update pipeline. The local-first architecture—powered by Raspbian OS, Mosquitto, and Tailscale—proves effective in delivering cloud-independent IoT management for privacy-conscious environments.

The successful integration of secure remote access (via Tailscale), intuitive UI design, and streamlined firmware deployment highlights the project's real-world readiness. While certain areas like auto-discovery optimization and mobile dashboard UX could benefit from further enhancements, the system in its current state already offers a comprehensive and secure solution for managing smart devices in a LAN-based environment.

With further development, Raspberry House could be expanded into a scalable private smart home controller or deployed in industrial and educational contexts requiring secure and offline-friendly IoT solutions.

CHAPTER 7
CONCLUSION AND FUTURE SCOPE

Chapter 1: Introduction

This chapter introduces the Raspberry Home project, which seeks to build a secure, scalable, and private network infrastructure tailored for small-scale IoT devices. It defines the core objectives of ensuring localized control, robust encryption, and seamless device interoperability within a LAN environment using affordable hardware like Raspberry Pi and ESP32. By identifying key limitations in current IoT ecosystems—such as reliance on cloud services, latency, and privacy risks—this project emphasizes the need for a local-first approach to smart device connectivity.

The introduction also highlights the broader significance of this work in the context of edge computing and privacy-respecting IoT solutions. By proposing a framework that empowers users to retain full control over their devices and data, Raspberry Home positions itself at the intersection of cybersecurity, embedded systems, and practical home automation.

Chapter 2: Review of Literature

This chapter reviews the current research and practices in the domain of IoT networking, MQTT protocols, TLS encryption, and over-the-air (OTA) firmware updates. By surveying academic articles, white papers, and open-source implementations, we identify trends and technological gaps, particularly around issues of centralized data dependency, lack of robust encryption in low-cost systems, and firmware update security.

The literature review thus sets the context for Raspberry Home by illustrating the necessity of a decentralized, encrypted, and maintainable architecture. It also reveals a clear lack of integration between OTA mechanisms and secure MQTT implementations for low-powered devices, paving the way for this project's novel contributions.

.

Chapter 3: Detailed Designing of Hardware Components

This chapter outlines the design and selection of hardware components fundamental to Raspberry Home Key elements include the Raspberry Pi (acting as the central broker and controller), ESP32 microcontrollers (as client nodes), and optional peripherals such as relays, sensors, and actuators. Each component was selected for its cost-effectiveness, open-source support, and compatibility with secure protocols.

We also discuss the networking configuration and power considerations necessary for long-term deployment. The use of a local MQTT broker over TLS, hosted on Raspberry Pi, ensures that communication between devices remains encrypted and locally confined, minimizing security risks.

Chapter 4: Detailed Integration of Software Tools

Software integration forms the core of Raspberry Home, where open-source tools are woven together to build a secure and responsive system. This chapter details the deployment of Mosquitto MQTT broker with TLS on Raspberry Pi, configuration of ESP32 nodes using the Arduino IDE, and implementation of OTA updates via HTTPS. Additional tools such as OpenSSL, system, and Nginx (for secure web dashboard hosting) are discussed in depth.

The software stack is designed to prioritize modularity and security while remaining lightweight and responsive. By abstracting communication through encrypted MQTT topics and enabling firmware updates over local HTTPS, the system minimizes cloud dependencies while ensuring ease of maintenance.

Chapter 5: Methodology and System Flow Chart

This chapter presents the systematic development process of Raspberry Home, broken into key phases: requirement analysis, hardware setup, secure communication implementation, node programming, dashboard configuration, and OTA testing. Each step was executed iteratively with frequent validation to maintain security and reliability.

A detailed flowchart illustrates the architecture, beginning from the user interface and command input, through MQTT topic publication, to ESP32 command execution and telemetry reporting. The flowchart also highlights the secure TLS handshake, authentication, and OTA update processes, offering a transparent view of the system's real-time behavior.

Chapter 6: Results and Discussion

The results chapter analyses the system's performance in terms of responsiveness, security, and scalability. Benchmarks were conducted for MQTT latency, OTA update time, and TLS handshake duration. The system demonstrated robust performance with minimal latency and successful OTA updates under varied network conditions.

Security validation tests confirmed encryption integrity, resistance to unauthorized device access, and proper client-broker authentication. While the system operated reliably under load, challenges such as certificate renewal complexity and memory constraints on ESP32 were identified, suggesting areas for future optimization

Overall Conclusion

The Raspberry Home project successfully demonstrates a localized, secure, and update-friendly network for small IoT devices. By combining hardware affordability with best practices in software security and design modularity, it addresses crucial pain points in typical consumer-grade IoT deployments.

This project not only shows the feasibility of a local-first IoT approach but also encourages further exploration into secure firmware distribution, scalable topology management, and broader device compatibility. It stands as a promising framework for developers seeking to build trustworthy and autonomous smart environments without sacrificing privacy or control.

REFERENCES

1. W. Fei, H. Ohno, and S. Sampalli, "Design and Implementation of Raspberry Home : An IoT Security Framework," in Proc. 2020 IEEE Int. Conf. on Internet of Things and Intelligence System (IoTaIS), Bali, Indonesia, 2020, pp. 1–7, doi: 10.1109/IoTaIS50849.2021.9359722.

2. J. Zhang, "Research on Key Technology of VPN Protocol Recognition," in Proc. 2021 IEEE Int. Conf. of Safety Produce Informatization (IICSPI), Chongqing, China, 2021, pp. 161–166, doi: 10.1109/IICSPI.2018.8690388.

3. K. Zandberg, K. Schleiser, F. Acosta, H. Tschofenig, and E. Baccelli, "Secure Firmware Updates for Constrained IoT Devices Using Open Standards: A Reality Check," IEEE Access, vol. 7, pp. 71907–71918, 2022, doi: 10.1109/ACCESS.2019.2919760.

4. P. Shen, Y. Qi, W. Yu, J. Fan, and F. Li, "OTA Measurement for IoT Wireless Device Performance Evaluation: Challenges and Solutions," IEEE Internet of Things J., vol. 5, no. 3, pp. 2031–2045, 2022, doi: 10.1109/JIOT.2018.2868787.

5. K. A. Arunkumar, K. Kriti, A. Das, and B. Bhattacharyya, "Wireless Speakers Using WiFi and IoT," in Proc. 2020 Int. Conf. on Vision Towards Emerging Trends in Communication and Networking (ViTECoN), Vellore, India, 2020, pp. 1–6, doi: 10.1109/ViTECoN.2019.8899571.

6. S. Kent and K. Seo, "Security Architecture for the Internet Protocol," RFC 4301, Dec. 2005. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4301.html

7. Schneier, Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd ed. New York, NY, USA: Wiley, 1995.

8. Raspberry Pi Foundation, "Raspberry Pi Documentation," [Online]. Available: https://www.raspberrypi.org/documentation

9. E. Rescorla, "HTTP Over TLS," RFC 2818, May 2000. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2818.html

APPENDIX

Format for Project details with relevance to environment, safety, ethics, cost and Mapping with Pos and PSO with justification

SI.NO	Title of project	Roll no of students	Faculty	Relevance to Environment	Relevance to Human safety	Relevance to Ethics	Cost (Hardware and Software cost)	Type (Application, Product, Research, Review)
1.	Raspberry Home-Secure Network for IoT devices	1604-21-735-91 1604-21-735-93 1604-21-735-93	Dr. Salma Fauzia	3	1	3	Rs.8000	IoT Security

MAPPING WITH POs AND PSOs

Details	Design a secure private IoT network using Raspberry Pi and ESP32, integrating MQTT over TLS for encrypted communication, local control, and OTA updates.
PO1	Apply knowledge of electronics, networking, and embedded systems to design and build a secure IoT architecture.
PO2	Identify and solve engineering problems related to secure device communication, remote access, and network isolation.
PO3	Design and configure secure communication protocols and services in real-world IoT scenarios.
PO4	Use tools and techniques for the design and implementation of complex systems involving software, hardware, and networking.
PO5	Conduct experiments and interpret data to validate system functionality and performance.
PO6	Design system-level architecture integrating multiple subsystems.
PO7	Create interfaces or methods for interaction between systems or users securely and efficiently.
PO8	Manage the project life cycle, from planning to execution and documentation.
PO9	Develop and apply testing protocols for firmware, network security, and end-to-end communication.
PO10	Stay current with technological developments in IoT, embedded systems, and cybersecurity.

Details	Design a secure private IoT network using Raspberry Pi and ESP32, integrating MQTT over TLS for encrypted communication, local control, and OTA updates.
PO11	Collaborate and communicate technical details with peers, mentors, and stakeholders.
PO12	Understand societal, ethical, and environmental implications of IoT security and privacy.

PSO1	Design and develop secure embedded systems and networked applications using microcontrollers and gateways.
PSO2	Develop practical skills in configuring and securing communication protocols like MQTT, HTTPS, and VPNs.
PSO3	Analyze and troubleshoot complex systems involving embedded software, network services, and cloud/remote tools.
PSO4	Demonstrate the ability to deploy scalable and secure IoT solutions applicable to real-world environments.

PROGRAM OUTCOMES

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.