Smart Anti-Theft System Documentation

1. Introduction

1.1 Overview

The Smart Anti-Theft System is designed to detect unauthorized access and send real-time alerts to users or security personnel. The system uses ESP32 as the core microcontroller and integrates communication protocols such as MQTT or HTTP to facilitate real-time notifications. Additionally, the system provides a user-friendly interface for monitoring and controlling the security setup.

1.2 Objectives

- Develop a robust system architecture for the Smart Anti-Theft System.
- Implement sensors and other hardware components for unauthorized access detection.
- Integrate communication protocols to send real-time alerts.
- Design and develop a user interface for monitoring and controlling the system.
- Conduct thorough testing and debugging to ensure system reliability and effectiveness.

2. System Architecture

2.1 Hardware Components

- **ESP32 Microcontroller:** The brain of the system, responsible for handling sensor data and communication.
- **PIR Sensor:** Detects motion or unauthorized access.
- **Buzzer:** Sounds an alarm when unauthorized access is detected.
- **LED:** Indicates system status (active, inactive, alarm triggered).

2.2 Circuit Diagram

- **PIR Sensor:**
    - **VCC:** Connected to 3.3V of ESP32.
    - **GND:** Connected to GND of ESP32.
    - **OUT:** Connected to GPIO 33 of ESP32.
- **Buzzer:**
    - **Positive:** Connected to GPIO 25 of ESP32.
    - **Negative:** Connected to GND of ESP32.
- **LED:**

    - **Positive:** Connected to GPIO 26 of ESP32.
    - **Negative:** Connected to GND of ESP32.

3. Software Design

3.1 Libraries Used

- **WiFi.h:** For connecting to Wi-Fi.
- **PubSubClient.h:** For MQTT communication.
- **HTTPClient.h:** For HTTP communication (if using HTTP protocol).

3.2 MQTT Integration

- **Broker:** `broker.hivemq.com` (or any other MQTT broker).
- **Port:** `1883`.
- **MQTT Topic:** `home/security/alert`.

3.3 HTTP Integration

- **Server URL:** `http://your-server.com/api/alert`.
- **Payload:** JSON data containing alert information.

3.4 Code Structure

3.4.1 Setup Function

- Initialize Wi-Fi connection.
- Connect to MQTT broker or HTTP server.

3.4.2 Loop Function

- Monitor PIR sensor for motion detection.
- Trigger buzzer and LED on motion detection.
- Send alert via MQTT or HTTP.

3.4.3 Reconnect Function (for MQTT)

- Handle reconnection to the MQTT broker if the connection is lost.

3.5 Example Code

Refer to the example codes provided earlier for both MQTT and HTTP integrations.

4. Communication Protocols

4.1 MQTT Protocol

- **Publish Topic:** Sends alerts when unauthorized access is detected.
- **Subscribe Topic:** Optional for receiving commands from the user interface (e.g., activate or deactivate the system).

4.2 HTTP Protocol

- **POST Request:** Sends alerts to the web server.
- **GET Request:** Fetches system status from the server.

5. User Interface

5.1 Web Dashboard

- **Technology:** HTML, CSS, JavaScript.
- **Features:**
    - Display system status.
    - Activate or deactivate the system.
    - Receive and display alerts.

5.2 Mobile App (Optional)

- **Framework:** React Native, Flutter, or Blynk.
- **Features:**
    - Real-time alerts.
    - Control the system remotely.

6. Testing and Debugging

6.1 Functional Testing

- **Motion Detection:** Verify the system detects motion accurately.
- **Alert Notification:** Test the system's ability to send alerts via MQTT or HTTP.
- **UI Interaction:** Ensure the user interface updates and interacts with the system correctly.

6.2 Edge Case Testing

- **Network Failure:** Test the system's behavior during network outages.

- **Multiple Alerts:** Handle scenarios where multiple alerts are triggered rapidly.
- **System Reset:** Ensure the system resumes normal operation after a reset.

6.3 Debugging

- **Serial Monitor:** Use the Serial Monitor to output debugging information.
- **MQTT Clients:** Use MQTT clients like MQTT.fx or HiveMQ WebSocket Client for testing.
- **HTTP Tools:** Use Postman or similar tools to test HTTP requests.

7. Deployment

7.1 Hardware Deployment

- Connect the ESP32 to the actual hardware components (PIR sensor, buzzer, LED).
- Deploy the system in the intended physical environment.

7.2 Software Deployment

- **MQTT Deployment:** Deploy the code on ESP32, ensuring a stable connection to the MQTT broker.
- **HTTP Deployment:** Ensure the web server is up and running to handle alerts.

8. Maintenance and Future Enhancements

8.1 Maintenance

- Regularly update the firmware for bug fixes and performance improvements.
- Ensure the MQTT broker or HTTP server remains operational.

8.2 Future Enhancements

- **Additional Sensors:** Integrate more sensors like door/window sensors or cameras.
- **Cloud Integration:** Store alert data in the cloud for historical analysis.
- **Mobile Notifications:** Implement push notifications for real-time alerts on mobile devices.

9. Conclusion

The Smart Anti-Theft System is a reliable and scalable solution for enhancing security in residential or commercial environments. By integrating robust communication protocols and providing a user-friendly interface, the system ensures real-time alerts and effective monitoring.

Further enhancements can be made to scale the system's capabilities and improve its performance.