**Systemspecifikation för Proof of Concept (PoC): IoT-baserad Rörelsedetektering med Säker mTLS MQTT-Kommunikation**

**1. Inledning**

Denna Proof of Concept (PoC) beskriver en IoT-lösning för säker och tillförlitlig rörelsedetektering. Målet är att demonstrera en komplett och robust arkitektur där data från en rörelsesensor (HC-SR501) skickas säkert via Wi-Fi till en lokal server byggd på en Raspberry Pi. Systemet utmärker sig genom användningen av **mTLS (Mutual Transport Layer Security) över MQTT** för att säkerställa ömsesidig autentisering mellan enhet och server.

Lösningen baseras på arkitekturen i git-arkivet **Ameleas/Secure\_Raspberry\_Pi\_IoT\_Server** och utnyttjar beprövade komponenter som Node-RED, InfluxDB och Grafana för databehandling, lagring och visualisering. PoC:n är utformad för att bevisa systemets funktion, säkerhet och skalbarhet för framtida kommersiella utrullningar inom övervaknings- eller fastighetsautomation.

**2. Systemarkitektur**

Systemet är uppdelat i tre huvudskikt: sensorenhet, kommunikationsprotokoll och backend-server.

**2.1 Enhetsarkitektur (Klientsidan)**

* **Mikrokontroller:** **ESP32** (Wi-Fi-kapabel). Den hanterar logik, sensoravläsning och den krypterade kommunikationen.
* **Sensor:** **HC-SR501 PIR-rörelsesensor**. Genererar en digital signal vid detekterad rörelse.
* **Firmware/Mjukvara:** Implementerar MQTT-klienten med stöd för mTLS. ESP32 lagrar sitt unika klientcertifikat och den betrodda Certificate Authority-kedjan (CA) för att autentisera servern.
* **Funktion:** När rörelse detekteras, initierar ESP32 en säker mTLS-anslutning och publicerar ett MQTT-meddelande (t.ex. {"status": "motion\_detected"}) till servern.

**2.2 Backend/Serverarkitektur**

Backend-systemet körs på en **Raspberry Pi** och använder en containeriserad miljö (Docker/Docker Compose, i linje med det angivna git-arkivet) för enkel utrullning och hantering.

| **Komponent** | **Funktion** |
| --- | --- |
| **MQTT Broker** | Hanterar meddelanden från ESP32-enheter. Konfigurerad att **kräva mTLS** för alla klientanslutningar, vilket säkerställer att endast certifikatsignerade enheter kan publicera data. |
| **Node-RED** | Fungerar som en *middleware* för bearbetning av dataflöden. Prenumererar på MQTT-meddelanden, filtrerar/transformerar datat och skickar det vidare till databasen. Kan även hantera larm och aviseringar. |
| **InfluxDB** | En tidsseriesdatabas optimerad för att lagra IoT-data (rörelsehändelser med tidsstämplar). |
| **Grafana** | Visualiseringsverktyg som läser data från InfluxDB. Används för att presentera rörelsehändelser i realtid, historisk översikt och hantera varningsnivåer. |

**2.3 Kommunikationsprotokoll**

* **Wi-Fi:** Används för den fysiska anslutningen mellan ESP32 och det lokala nätverket (LAN) där Raspberry Pi-servern finns.
* **MQTT (Message Queuing Telemetry Transport):** Ett lightweight-protokoll för meddelandehantering, idealiskt för IoT-enheter med begränsade resurser.
* **mTLS (Mutual Transport Layer Security):** Säkerhetslager som omsluter MQTT-trafiken (kallas även MQTTS).

**3. Säkerhetsåtgärder och Kryptering**

Säkerhet är kärnan i denna PoC, implementerad genom mTLS (ömsesidig autentisering).

* **Ömsesidig Autentisering (mTLS):** Till skillnad från standard-TLS, där endast klienten autentiserar servern, kräver mTLS att **både** klienten (ESP32) och servern (MQTT-brokern) autentiserar varandra med X.509-certifikat. Detta garanterar:
  1. **Serverns Integritet:** ESP32 vet att den pratar med den legitima Raspberry Pi-servern.
  2. **Klientens Integritet (Auktorisering):** Servern accepterar endast anslutningar från legitima, certifikatsignerade ESP32-enheter, vilket förhindrar obehöriga enheter från att injicera falsk data.
* **Infrastruktur för PKI (Public Key Infrastructure):** En privat Certificate Authority (CA) används för att utfärda och signera alla klient- och servercertifikat. Detta hanteras på Raspberry Pi-servern (enligt det angivna git-arkivets metoder).
* **Datakryptering:** All data som skickas mellan ESP32 och servern är fullständigt krypterad med TLS/SSL, vilket skyddar mot avlyssning och man-in-the-middle-attacker.
* **Certifikathantering på Enheten:** ESP32 är konfigurerad för att säkert lagra de nödvändiga certifikatfilerna och privata nycklarna, vilket är avgörande för att upprätthålla mTLS-säkerheten.

**4. Skalbarhet och Framtida Utveckling**

Lösningen är designad med skalbarhet i åtanke, vilket möjliggör enkel utökning av antalet sensorer och migrering till en mer robust infrastruktur.

* **Skalbarhet på Klientsidan:**
  + **MQTT:s Publish/Subscribe-modell:** Systemet är naturligt skalbart. Varje ny ESP32/HC-SR501-enhet kan läggas till genom att generera ett unikt mTLS-certifikat och ansluta till samma MQTT-ämne. Brokern hanterar enkelt tusentals samtidiga anslutningar.
* **Skalbarhet på Serversidan (Migration):**
  + **Molnredo Arkitektur:** Genom att använda etablerade, molnkompatibla komponenter (MQTT, InfluxDB, Grafana) kan hela backend-stacken enkelt migreras från Raspberry Pi till en robust molnplattform (t.ex. AWS, Azure eller GCP) när prestanda- eller skakrav överstiger Raspberry Pi:s kapacitet.
  + **Separation av Tjänster:** Komponenten är separerade, vilket underlättar *horisontell skalning*. Till exempel kan flera Node-RED-instanser läggas till bakom en lastbalanserare för att hantera högre datavolymer.

**5. Driftsättning och Underhåll**

* **Systemunderhåll:** Användningen av containerisering (Docker) förenklar systemunderhåll, uppdateringar och säkerhetskopiering.
* **Firmwareuppdateringar (FOTA - Framtida):** För en framtida produktionsmiljö kan ESP32-firmware uppgraderas för att inkludera FOTA-funktionalitet, vilket möjliggör säkra fjärruppdateringar av enhetens mjukvara utan att kompromissa med mTLS-konfigurationen.
* **Certifikatförnyelse:** En process för regelbunden förnyelse av mTLS-certifikat måste etableras för att upprätthålla en hög säkerhetsnivå. Detta kan automatiseras via Node-RED eller ett separat skript.