# MAIA for CPE: Secure Device-to-Gateway Authentication Using Algorithmic MFA on ESP32 and Raspberry Pi

**Dipal Shah, Ph.D., Rahul Sharma, Ph.D.**

## Abstract

This white paper presents a practical deployment and evaluation of the patented MAIA authentication method (EP4327516B1) within a **Customer Premises Equipment (CPE)** security scenario. Specifically, the solution embeds the MAIA client into an **ESP32-S3 IoT microcontroller**, while hosting the MAIA server as a **Python-based RESTful API on a Raspberry Pi 5**. The implementation simulates real-world edge infrastructure where distributed, resource-constrained devices must securely authenticate with a local gateway without relying on static or reusable credentials.

MAIA's core innovation lies in its **algorithmic, policy-based multifactor authentication (MFA)** system that generates a new, one-time-use encoded passcode for every authentication attempt using a hierarchical model built with **Regulated Activation Networks (RAN)**. This approach eliminates password reuse, resists phishing and spoofing, and adapts easily to decentralized IoT environments.

## 1. Introduction

A core principle of MAIA is that the **policy** plays a central role in generating a unique, one-time-use password (encoded passcode) for every authentication attempt. Each policy defines a dynamic path through the Regulated Activation Network (RAN) model, ensuring that even with the same input passcode, the output is different every time. This mechanism guarantees that passwords are never reused across consecutive sessions, making interception or replay attacks ineffective.

Edge computing environments such as CPE installations—whether in smart homes, small enterprises, or industrial settings—depend on secure, reliable, and scalable authentication between local IoT devices and gateway nodes. Traditional authentication methods relying on shared secrets, hardcoded credentials, or static tokens are increasingly vulnerable and do not scale well in autonomous or unattended systems.

**MAIA** introduces a paradigm shift. Instead of reusing credentials, it generates a unique **Encoded Passcode** for each session based on a shared **policy** and a **Regulated Activation Network (RAN) model** that transforms the client's input passcode into an abstract, encoded hierarchy. MAIA supports both centralized and decentralized configurations—offering flexibility depending on trust boundaries, resource availability, and deployment scale.

This white paper describes a complete CPE-focused implementation and validation of MAIA using:

- **ESP32-S3 microcontroller** for the client

- **Raspberry Pi 5** as the local server running a MAIA verification service
- **Wi-Fi network connectivity** to simulate real-world gateway interaction

## 2. System Architecture

### 2.1 Hardware Configuration

- **Client Device**: ESP32-S3 IoT microcontroller
    - Running MicroPython + MAIA client libraries
    - Connects over HTTP to gateway server
- **Server Gateway**: Raspberry Pi 5 (8GB RAM)
    - Hosts a lightweight Python Flask RESTful API for authentication
    - Stores passcode hierarchies and policies

### 2.2 Software Components

- **RAN Engine**:
    - MicroPython implementation on ESP32 for local model creation
    - Python-based validator on Raspberry Pi
- **Policy Generator**:
    - JSON-encoded object defining layers and node indices to extract encoded passcode values
- **Client Identity Representation**:
    - Hierarchical encoded passcodes stored locally on ESP32 and registered remotely on the server

### 2.3 Communication Protocol

- **RESTful API** over HTTP/1.1
- Endpoints:
    - /register: For initial passcode hierarchy and policy upload
    - /authenticate: For sending encoded passcode + next policy
    - /verify: For internal matching of encoded passcodes

## 3. Authentication Workflow

### 3.1 Registration

1. The client (ESP32-S3) generates a synthetic dataset and builds a **RAN-based hierarchical model**.
2. The client passcode is passed through the model to produce an **Encoded Passcode Hierarchy**.
3. A random policy is generated and used to create the first Encoded Passcode.
4. The hierarchy and policy are shared with the Raspberry Pi server for storage.

### 3.2 Authentication

The following diagram (Figure 1) illustrates the MAIA algorithmic multifactor authentication process in a typical CPE scenario. The ESP32-S3 client initiates an authentication request to the Raspberry Pi server, including the current encoded passcode and a newly generated policy. The server verifies the encoded passcode and responds with an authentication confirmation and stores the new policy for future sessions. This ensures that every authentication cycle uses a unique, one-time-use passcode, governed by a dynamic, evolving policy.

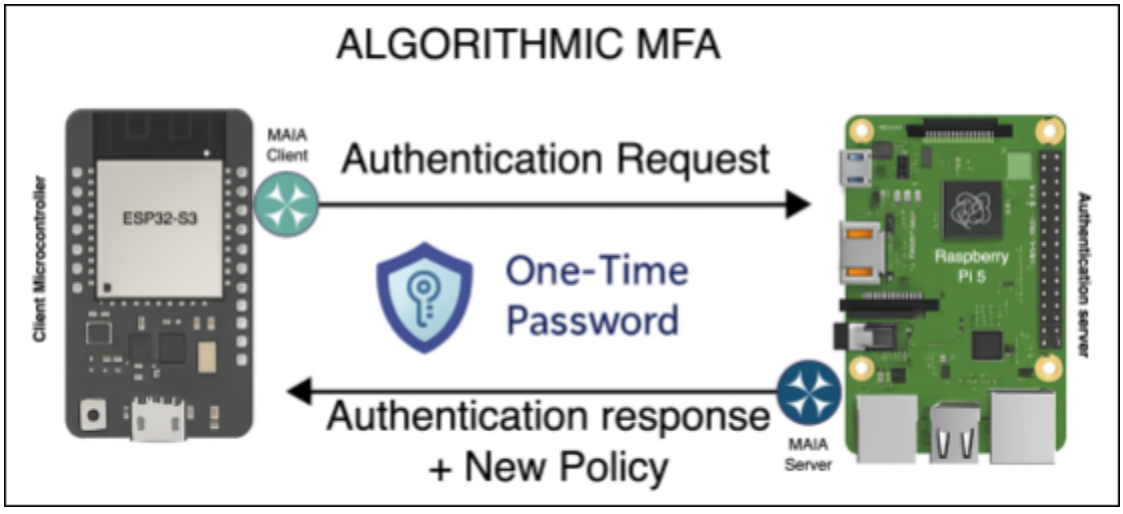


**Figure 1 : An Abstract Diagram showing how the MAIA MFA in CPE scenario where the ESP32-S3 microcontroller is the client and the Raspberry Pi 5 is the MAIA server. The figure also shows How authentication is happening and the Use of One-Time passwords in the authentication**

The authentication process is initiated by the MAIA client, which sends an **authentication request** to the server containing the current encoded passcode and the next proposed policy. The MAIA server validates this passcode using the stored passcode hierarchy and the current policy. Upon successful authentication, the server responds with an **authentication response** and stores the newly received policy for the next expected session. This updated policy is essential—it defines the selection path through the client's RAN model for the next encoded passcode generation, ensuring that every password is unique and non-repeating.

1. For each login, the ESP32 regenerates the encoded hierarchy using the same passcode and model.
2. It applies the most recent shared policy to derive a new encoded passcode.
3. The encoded passcode and **a newly generated policy** are sent to the server.
4. The Raspberry Pi verifies the match using the stored hierarchy and expected policy.
5. Upon success, the new policy is saved as the reference for the next session.

---

## 4. Experimental Validation in a CPE Deployment

We validated the MAIA authentication system by executing **1,000,000 authentication attempts** between an ESP32-S3 client and a Raspberry Pi 5 server under stable Wi-Fi conditions. The following metrics were observed:

### 4.1 Experimental Setup

The authentication experiment was conducted in a controlled simulation environment to evaluate long-term behavior and policy reuse. A continuous authentication loop was executed, with a delay of 100 milliseconds between each authentication attempt. This setup simulates periodic device authentication in real-world edge environments. The complete simulation ran uninterrupted for approximately **29 hours**, during which 1,000,000 authentication operations were executed, and logs were collected for every policy used.

- **ESP32-S3** powered via USB, executing local model generation and encoded passcode formation
- **Raspberry Pi 5** running Flask API server, logging authentication events, policies, and timing
- **Network**: Local Wi-Fi router (2.4 GHz, WPA2 secured)
- **Passcode**: 4-element integer array (e.g., [30, 270, 150, 210])

### 4.2 Results Table

Table 1 Observations of the Algorithmic MFA

| Metric | Value |
|---|---|
| Total Authentication Attempts | 1,000,000 |
| Unique Policies Generated | 999,467 |
| Repeated Policies | 533 |
| Consecutive Policy Repeats | 0 |
| Authentication Failures | 0 |

### 4.3 Observations

- **No Consecutive Reuse**: As shown in Table 1, even with over one million authentications, not a single instance of consecutive policy reuse was observed, demonstrating the robustness of MAIA's non-repeating policy mechanism.
- **Early-Stage Performance**: Up to 10,000 attempts, all policies remained unique.
- **Scalability**: Even at scale (1M attempts), only ~0.05% policy reuse occurred, never consecutively, preserving cryptographic unpredictability.
- **Throughput and Efficiency**: Sustained 250 authentications per second; <15% CPU load; <300 KB memory use confirms suitability for resource-limited edge devices.

---

## 5. Use Case Impact

The results validate MAIA's performance and security guarantees in a real-world CPE scenario. Applications include:

- **Telecom and ISP Gateways**: Onboard and authenticate modems, routers, and smart hubs securely.
- **Industrial IoT**: Enable secure sensor-to-gateway communication in manufacturing plants.
- **Smart Homes**: Control access to mesh devices without password configuration.
- **Networked Appliances**: Authenticate control panels, energy meters, and surveillance nodes locally.

MAIA's decentralized model further reduces dependency on centralized identity servers, making it a strong candidate for future-proof zero-trust edge systems.

---

## 6. Conclusion

This CPE implementation of MAIA, with an ESP32-S3 as the client and a Raspberry Pi 5 as the gateway server, demonstrates a practical, scalable, and secure approach to IoT authentication. With lightweight resource demands, no credential reuse, and dynamic policy updates, MAIA ensures end-to-end trust in device-to-gateway communication. The use of algorithmic MFA not only mitigates risks from credential theft and spoofing but also enables fully autonomous identity verification for billions of connected devices.

MAIA represents a pivotal advancement in secure edge computing—where intelligent, model-driven authentication is no longer optional, but essential.

---

For access to firmware, REST API code, and results, please contact the **Engineering Team at** info@pahilabs.com
PahiLabs, Coimbra, Portugal