Prevention of Man-in-the-Middle Attacks in Offline Embedded Systems

Research Project Documentation

December 17, 2025

Contents

1 Introduction

The CIA triad—confidentiality, integrity, and availability—is commonly used to evaluate security threats, and man-in-the-middle (MitM) attacks pose serious risks to all three aspects in IoT systems. MitM attacks are broadly classified into passive and active attacks. Passive MitM attacks involve silently eavesdropping on communications to obtain sensitive information without altering the data, while active MitM attacks involve intercepting and modifying communications through techniques such as impersonation, spoofing, and data tampering.

A typical MitM attack in an IoT environment follows three main stages: reconnaissance, interception, and data modification. During reconnaissance, attackers scan the network to identify devices, IP addresses, and vulnerabilities using tools such as Nmap or Wireshark. In the interception phase, techniques like ARP spoofing, DNS spoofing, rogue access points, or compromised MQTT brokers are used to capture traffic. Finally, attackers may modify, inject, or manipulate the intercepted data before forwarding it to the intended recipient.

Detecting and preventing MitM attacks in IoT systems is challenging due to several factors. IoT devices are typically resource-constrained, limiting their ability to implement strong cryptographic and monitoring mechanisms. The heterogeneity of IoT devices, operating systems, and communication protocols makes it difficult to apply uniform security measures. Additionally, while encryption is essential for protection, it introduces computational and latency overhead, creating a trade-off between security and performance in IoT environments.

2 Project Concept and Initial Discussion

2.1 Core Idea

The research project focuses on prevention of man-in-the-middle attacks for offline systems, with particular emphasis on embedded systems. The concept put forth involves a server S and clients A, B, C, with encryption helpers like TPM, VPUF and so on.

The architecture consists of three main components:

2.1.1 Bootup Sequence with Manual Supervision

A bootup sequence is done under manual supervision, which will be done in a mutex fashion. At a time, only one client can communicate with server and no other client, not even a neutral can access the medium. In this phase, all info regarding encryption protocols and standards to be chosen along with characteristics of each client is sent. It is done with manual intervention to avoid the case wherein an attacker D can pretend as a neutral client C by attacking the info sent by C to the medium.

2.1.2 Communication Protocol

Suppose A wants to communicate with C, A sends the communication info (it wants to communicate with C) to the server. The server then uses info of C to craft a game which is not parallelizable to prevent brute force attacks and the games can only be solved by C to a high degree, and C wins the game and others lose. After this, the server will share a nonce key of C and using that key, the message will get encrypted and sent through the

medium. To improve encryption, salting of multiple nonce of C can be done on packets of A so that A can send it comfortably and no other attacker can crack the packets and get the data.

2.1.3 Watchlist and Baiting Mechanism

A watchlist will keep track of suspects that try to crack games that are not intended for them. The system will also use baiting games that no other device is supposed to solve at random intervals which will confirm suspicion and eventually, the device will be denied from communicating if a particular threshold exceeds.

2.2 Week 1 Discussion Summary

The team discussed with the guide about designing a secure communication framework to prevent MITM attacks in offline embedded systems. The key idea is that although the system operates without any external authentication servers, a central server within the isolated network can still coordinate device interactions and ensure secure message exchange. The discussion also covered three core aspects of the proposed approach: a secure one-time supervised setup for each client, server-controlled authentication using device-specific challenges, and a mechanism to detect and isolate suspicious devices.

2.3 Key Challenges Identified

Boot-time attack prevention: Prevent attackers from spoofing legitimate clients during initialization.
Challenge design: Ensure challenges are solvable only by the intended client.

3 Understanding Network Communication and MitM Attacks

3.1 How Actual Communication Occurs in Networks

At the most basic level, communication between two devices follows a specific flow. Device A needs to talk to device B. To do that, A must know who B is (identity), where B is (address), and how to talk to B (protocol). In Ethernet/IP networks, this involves IP address and MAC address. In embedded buses like CAN, SPI, or I²C, it involves node ID or bus address. In industrial networks, it's device ID plus controller route.

Before sending real data, devices establish a session where they agree on parameters like encryption, keys, and counters. They also verify identities, either explicitly or implicitly. This is similar to a TLS handshake on the Internet, or secure CAN key exchange in automotive ECUs, or authenticated command handshake in PLCs.

Once the session is established, actual data is transmitted. Packets or frames flow through a medium, and the medium itself is never trusted. Encryption plus integrity checks protect the data during transmission.

3.2 Where Man-in-the-Middle Attacks Occur

A MITM attack happens when an attacker controls or monitors the communication medium. Common MITM scenarios include when an attacker sits on the same Wi-Fi network, when there's a compromised switch or router inside a LAN, when there's physical access to a CAN bus or Ethernet line, or when there's malicious firmware inside a gateway ECU.

What the attacker does is intercept messages between A and B, pretend to be B when talking to A, pretend to be A when talking to B, and optionally modify messages silently. An important point here is that encryption alone does not prevent MITM if identities are not verified.

3.3 How MITM Actually Succeeds

In a scenario with no authentication, A sends "Hello B", but attacker D intercepts. D responds "Hello A, I am B", and A trusts D and sends data. D then forwards or alters data to real B. Neither A nor B realizes someone is in the middle.

3.4 Prevention Principles

All real-world defenses follow three fundamental rules. First, strong identity verification where each device must prove who it is using something an attacker cannot copy. Examples include cryptographic private keys, hardware secure elements like TPM or SE, or Physical Unclonable Functions (PUFs). HTTPS prevents MITM because the server proves its identity using a certificate signed by a trusted CA.

Second, freshness or anti-replay where each session must be unique. Techniques include nonces, timestamps, and counters. TLS uses random nonces so old intercepted handshakes can't be replayed.

Third, integrity protection where messages must be tamper-evident. Techniques include Message Authentication Codes (MACs) and authenticated encryption like AES-GCM or ChaCha20-Poly1305. Automotive Secure CAN attaches MACs to each message.

4 ARP Cache Poisoning

4.1 Understanding ARP and ARP Cache

ARP (Address Resolution Protocol) is used in a local network (LAN) to map IP address to MAC address. For example, your PC knows the IP of the router is 192.168.1.1, but to send Ethernet frames, it needs the MAC address. So it sends "Who has 192.168.1.1?" and the router replies "192.168.1.1 is at AA:BB:CC:DD:EE:FF". This mapping is stored in a table called the ARP cache.

An important ARP weakness is that ARP has no authentication. Any device can send an ARP reply, even if nobody asked for it. The receiver will usually trust and update its ARP cache. This is the core vulnerability.

4.2 What is ARP Cache Poisoning

ARP cache poisoning occurs when an attacker sends forged ARP replies to victims, causing them to store incorrect IP to MAC mappings. In simple terms, the attacker

convinces devices that their MAC address belongs to someone else's IP address.

4.3 How ARP Poisoning Leads to MitM

Consider three devices in the same LAN: Victim (User's PC), Router (Default gateway), and Attacker (Malicious device). In the normal state, Victim's ARP cache shows Router IP maps to Router MAC, and Router's ARP cache shows Victim IP maps to Victim MAC. Traffic flows directly between Victim and Router.

The attacker sends two forged ARP messages. First, to the Victim: "I am the router (192.168.1.1), my MAC address is Attacker MAC". Second, to the Router: "I am the victim (Victim IP), my MAC address is Attacker MAC".

Now ARP cache poisoning occurs. The Victim's ARP cache shows Router IP maps to Attacker MAC, and Router's ARP cache shows Victim IP maps to Attacker MAC. Both devices unknowingly send traffic to the attacker.

The Man-in-the-Middle is established. The traffic path becomes Victim to Attacker to Router and Router to Attacker to Victim. The attacker can now sniff packets including passwords, cookies, and sessions, modify data, drop packets causing DoS, or perform SSL stripping or DNS spoofing. Crucially, communication still works, so neither side suspects anything.

4.4 Why This Attack Works Easily

ARP is stateless, ARP replies are trusted blindly, there's no verification of sender identity, and it works only within the same broadcast domain (LAN).

5 Prevention Strategies for ARP Poisoning in Offline Systems

For the research project context, since we're dealing with an offline system (no Internet but still a local network), ARP poisoning is absolutely still possible because ARP is local-network only. The core principle for prevention is to remove ARP's "blind trust". ARP poisoning succeeds because any device can send ARP replies and devices accept them without verification. So prevention means either verify ARP messages or eliminate ARP dependence.

5.1 Static ARP Binding

Static ARP binding is the most practical approach for offline systems. You manually bind IP address to MAC address, and the OS refuses to update that entry dynamically. Offline networks usually have fixed nodes, known IPs, and stable topology, making it a perfect fit for static bindings.

For implementation, in Linux you use `sudo arp -s 192.168.1.1 AA:BB:CC:DD:EE:FF`. In embedded Linux, you hardcode ARP entries during boot or configure via init scripts. The effect is that forged ARP replies are ignored and ARP poisoning fails completely. The limitation is that it's not scalable for large, dynamic networks, but it's excellent for embedded or lab setups.

5.2 ARP Packet Validation

Instead of trusting ARP blindly, validate ARP packets. This can be done through ARP inspection logic in software. You monitor ARP traffic and verify sender IP-MAC consistency and that MAC belongs to known whitelist, then drop suspicious ARP replies.

Implementation can use a custom daemon using libpcap or raw sockets, which is lightweight for embedded systems. The checks to enforce are: if ARP reply is received and IP-MAC pair is not in trusted table, then drop it. This is similar to Dynamic ARP Inspection, but done in software.

5.3 Cryptographic Authentication Above ARP

Even if ARP is compromised, traffic remains secure. Use mutual authentication at transport or application layer with TLS with mutual certificates, pre-shared keys (PSK), or Message Authentication Codes (HMAC). The attacker can forward packets but cannot modify or decrypt them, making the MitM useless.

This aligns well with security-level assignment work: confidentiality through encryption, integrity through MAC, and authentication through mutual auth. Even under potential link-layer attacks, end-to-end cryptographic authentication ensures data integrity and confidentiality.

5.4 Network Segmentation

Using VLAN or physical isolation, you can separate control nodes, data nodes, and monitoring nodes. Even in offline systems, physical segmentation drastically reduces attack feasibility. For embedded systems, using a dedicated switch with no shared broadcast domain helps.

6 Refined System Architecture

6.1 Hierarchical Trust Model

The proposed system can be considered as a system of systems. An embedded system with multiple IoT devices has a set of sensors that will sense a particular activity, a set of actuators that will do a particular thing and so on. These can be classified based on their particular purpose. For each purpose, there will be a little controller that will handle everything and that controller is actually tied with the central server.

So there is a central server and for each particular function, there is a particular controller that will control the sensors, actuators and everything. There is proper authentication between the server and the controller so that the server trusts the controller entirely. The controller will actually verify each and every other device that gets plugged into the network.

For example, if a particular device tries to plug into the network, then it is the controller's duty to verify if it is an attacker or just a normal client. After that, it will send those packets to that particular port if it is not an attacker and if it is a genuine user. So it's kind of like a binding that is made at the subsystem level.

6.2 Trust Hierarchy Structure

The system implements a three-level trust hierarchy:

- **Central Server**: The top-level trust anchor
- **Trusted Subsystem Controllers**: A small, countable set (typically 10-12) whose identities are provisioned a priori at the server
- **Devices**: Large, scalable population of sensors and actuators that must authenticate to controllers

This architecture has introduced three critical security principles. First, a system-of-systems trust hierarchy where the central server connects to trusted subsystem controllers which connect to devices (sensors and actuators). This is much better than flat trust.

Second, delegated authentication where the server does not authenticate every device. Controllers act as local trust anchors and devices are authenticated within their functional domain. This scales well, matches embedded realities, and reduces attack surface.

Third, network admission control at subsystem level where the controller verifies each device before forwarding packets. This is the correct place to enforce trust: before routing, before application logic, and before meaningful communication.

6.3 Why This Architecture is Valid

This design resembles proven patterns used in various domains. In Industrial Control Systems, it's similar to cell or zone controllers. In Automotive systems, it's like gateway ECUs. In IoT Security, it's similar to edge trust brokers. In Zero Trust architectures, it's like Policy Enforcement Points (PEP). In Software-Defined Networking, it's like control-plane admission.

6.4 Identity and Authentication Mechanisms

Controllers are treated as trust anchors. The validity of the controller is already given to the server because the server will already host the identification details of the controller. This is normal because the number of controllers may be something like countable, like 10 or 12 or something, but the number of devices that actually send the data, that actually transfer or do other processing of the data, that is very large. The number of devices can scale very largely.

The devices are verified by the controller and the controller itself is verified by the server using the details that are provided to the server a priori. So before such a system even exists, some information regarding this particular controller is already sent to the server and based on this, the controller is already established. The controller is already considered as a trust anchor.

The system authenticates devices using device-specific or controller-specific features. The controller may have some PUF or may have some TPM or something and based on that and also some other values, we'll ensure that only the devices that have valid access to the controller, that are validly locked with the controller, can only access this controller. This detail is contained in the controller in a secure TPM-like module so that an attacker can only attack the system if it directly tampers with the information from the TPM module.

As mentioned before, the system ensures quarantine before authentication because the controllers will be authenticated in a pretty manual way and we'll ensure that no rogue controllers are present because the controllers themselves will verify whether the other devices are correct or not. The system is not relying on ARP or IP address for identity. It is relying on the cryptographic feature or physically unclonable features of a microcontroller for the identification.

6.5 Threat Coverage

Even if ARP is poisoned and an attacker sits between device and controller, they still cannot authenticate as a valid device, establish trusted bindings, or gain controller forwarding privileges. ARP spoofing becomes a transport nuisance, not a security breach. This is exactly the correct outcome.

The system is doing several things right. It is not trying to "fix ARP". It treats the LAN as hostile. It doesn't derive trust from addresses. It relies on hardware-backed identity. Many projects fail because they stop at "we'll detect ARP spoofing", but this approach goes two layers deeper.

7 DNS-Based Attacks

7.1 DNS Spoofing

DNS spoofing (also called DNS cache poisoning) is an attack in which an attacker injects false DNS records into a DNS resolver's cache so that a domain name resolves to a malicious IP address instead of the legitimate one. In simple terms, the attacker tricks the DNS system into lying about where a website actually lives.

Common ways attackers perform DNS spoofing include guessing DNS transaction IDs and racing legitimate responses, compromising a DNS server, exploiting unsecured Wi-Fi networks, or using ARP spoofing plus DNS response manipulation in local network attacks.

7.2 How DNS Spoofing Aids MitM

When a victim requests a domain like www.bank.com, if the attacker spoofs the DNS response, they return the attacker's IP instead of the real bank server. The victim then connects to the attacker believing it's the real website, and the MITM is established. The attacker can read credentials (usernames, passwords), modify transactions, inject malware, or proxy traffic to the real site to avoid suspicion.

DNS spoofing is effective for MITM because DNS is trusted by default, users rarely verify IP addresses, it works silently in the background, and can affect many users at once if a resolver is poisoned.

7.3 Applicability to the Research Project

DNS spoofing and DNS-based man-in-the-middle attacks are not applicable in the proposed system, as the architecture operates in a controlled, primarily offline environment with no dynamic DNS resolution. Since the attacker has no on-path presence or network

access to inject or manipulate DNS responses, such attacks are excluded from the threat model.

The system is primarily offline, meaning it's not connected to the internet, though a local network (LAN) will be there which will connect all the IoT devices and all the embedded devices. IP addresses cannot be statically assigned because some other devices might be added to the system, but you have to verify the integrity of the system as a whole by ensuring that there is no attackers in the system.

Even in a scenario where client A wants to send a message to client B and an attacker is present, if A is sending the message to B, then A will obviously encrypt using B's information. But before B's information is passed on to A, if an attacker acts as B and spoofs the information of B so that A will send the packets directly to the attacker, this is also a man-in-the-middle attack because both A and the attacker share the knowledge required to decrypt the message, so the attacker can decrypt the message and access it.

The system design eliminates the attacker's ability to be on-path or in control of the name-resolution channel; therefore DNS spoofing is outside the considered threat model. The real problem is not fixing ARP or DNS, but establishing identity before network trust is granted.

8 Session-Based Attacks

8.1 Session Hijacking Overview

Session hijacking is an attack where an attacker takes over an already authenticated session between a client and a server. The attacker does not need the password. They steal or manipulate the session identifier (cookie or token).

After login, the server issues a session ID (cookie, JWT, token) and the browser sends this ID with every request. Whoever has the session ID becomes the logged-in user. So if an attacker steals the session token, the attacker becomes that user.

8.2 Types of Session Hijacking

Sniffing is listening to network traffic to capture data. It happens on open Wi-Fi, compromised LAN, or poorly secured switches. If traffic is not encrypted (HTTP), cookies, session IDs, and credentials can be captured.

Side-jacking is a specific form of sniffing, popularized by tools like Firesheep. The key idea is that the login page uses HTTPS but subsequent requests use HTTP. So login is secure but the session cookie is sent in plaintext later. The attacker sniffs the cookie and performs side-jacking by stealing session cookies after login.

Evil Twin Attack is not session hijacking itself, but a setup. The attacker creates a fake Wi-Fi AP with the same SSID as the legitimate network and a stronger signal, so the victim connects. Once connected, the attacker controls the network and can sniff traffic, inject packets, or redirect requests. This leads to sniffing, MitM, and session hijacking.

8.3 Relationship with ARP and DNS Poisoning

ARP poisoning operates at Layer 2 (Data Link). The attacker lies about MAC to IP mapping, so the victim sends traffic to attacker instead of router. The result is that the attacker sits between victim and router providing full Man-in-the-Middle. This enables

sniffing, session hijacking, and traffic modification. ARP poisoning is a classic MitM enabler.

DNS poisoning operates at the Application layer. The attacker gives wrong IP for a domain so that bank.com resolves to attacker's IP. The result is that the victim connects to the attacker who can steal credentials, proxy traffic (MitM), or inject malware. DNS poisoning enables redirection-based MitM, not direct session stealing.

8.4 Attack Chain

Most real attacks follow this pattern: ARP Poisoning or Evil Twin or Rogue AP puts the attacker in a MitM position, then Sniffing observes traffic, and finally Session Hijacking takes control of the authenticated session.

So ARP poisoning or Evil twin puts attacker in the middle, sniffing observes traffic, and session hijacking takes control of authenticated session.

8.5 Applicability to Research Project

Session hijacking and related attacks such as side-jacking, evil twin, ARP poisoning, and DNS poisoning apply primarily to network-connected, session-based web systems. In an air-gapped system, these attacks are not applicable. Even in online systems, successful session hijacking typically results from improper application-level security (e.g., insecure cookie handling or lack of HTTPS), with network attacks merely acting as enablers rather than root causes.

The system being designed is air-gapped with no external network, no browser-server session, so there is no session to hijack. Session hijacking, side-jacking, evil twin, sniffing, and ARP/DNS attacks are not applicable.

9 SSL/TLS Hijacking

9.1 What SSL Hijacking Is

SSL hijacking (often discussed today as TLS hijacking) is a class of attacks where an attacker interferes with or compromises the secure TLS/SSL connection between a client (browser, app, IoT device) and a server, enabling a Man-in-the-Middle position.

It does not break TLS cryptography directly. Instead, it tricks one or both endpoints into accepting the attacker as a legitimate party. SSL hijacking means forcing or tricking a victim into establishing a TLS session with the attacker instead of the real server, while the attacker maintains another TLS session with the real server.

The result is that the client connects via TLS to the attacker who connects via TLS to the server. The attacker sees plaintext data, modifies traffic, and steals credentials or session cookies. This is a full MitM attack enabled via TLS misuse or downgrade, not cryptographic failure.

9.2 Mechanisms of SSL Hijacking

SSL Stripping is the most common conceptual example and is a downgrade attack. When a user types http://example.com, the server intends to redirect to https://example.com. The attacker intercepts the redirect and keeps the client on HTTP while talking HTTPS

with the server. The outcome is that the client thinks it's normal browsing while the attacker reads and modifies everything. This is prevented by HSTS.

Fake or rogue certificates occur when the attacker presents a fraudulent certificate and the victim device trusts it due to installed malicious root CA, corporate proxy CA, or compromised OS/browser trust store. This is common in malware-infected systems, enterprise TLS inspection proxies, and captive portals with bad practices.

Compromised or malicious Certificate Authority is rare but high impact. If a CA is compromised, coerced, or mis-issues certs, attackers can get legitimate certificates for domains they don't own. Historical examples include DigiNotar breach and CNNIC incidents.

Session hijacking after TLS termination can occur when TLS ends at load balancer, reverse proxy, or API gateway. If that component is compromised, the attacker can steal session cookies and replay them. This is post-TLS hijacking, still MitM-like.

DNS, ARP, or Evil Twin attacks don't break TLS by themselves, but they enable SSL hijacking. DNS spoofing redirects client to attacker, ARP poisoning positions attacker in network, and Evil Twin Wi-Fi forces traffic through attacker. Once positioned, attacker attempts SSL stripping, cert spoofing, or downgrade attacks. So SSL hijacking is often the payload while ARP/DNS are enablers.

9.3 Applicability to Research Project

Based on the system description with air-gapped system, TPM plus mTLS plus Attestation, the system is strongly resistant to SSL hijacking. There's no open network for positioning, mutual authentication prevents rogue endpoints, certificate pinning is likely in place, and there's no browser-based trust abuse.

If the system goes online, then SSL hijacking is only possible if you trust arbitrary CAs, have no certificate pinning, have no HSTS-like enforcement, or TLS terminates in untrusted middleware. With mTLS plus TPM-bound certs, SSL hijacking becomes practically infeasible.

10 Online Extension Architecture

10.1 Transitioning to Online Operation

When extending the system to an online deployment, an intermediate trusted server is introduced between IoT devices and the cloud. This server, being less resource-constrained, handles public-key infrastructure operations and validates cloud services using certificates issued by trusted public Certificate Authorities.

IoT devices communicate only with the trusted server using pre-established trust, thereby reducing exposure to DNS spoofing and man-in-the-middle attacks while maintaining scalability and security.

10.2 Trust Flow in Online Mode

The secure communication flow works as follows: IoT Device communicates with Trusted Server (Gateway) which communicates with Cloud.

For the link between IoT and Server, pre-installed public key or TPM-bound cert is used. For the link between Server and Cloud, public CA (Let's Encrypt, DigiCert, etc.) is used.

The system should not make IoT devices talk directly to public CAs. Once identity is authenticated above ARP, ARP spoofing becomes irrelevant. Many secure systems do not try to "fix ARP" at all. They accept that the network is hostile and that trust is established cryptographically.

10.3 Why This Architecture Prevents MitM

For the IoT to Server link, no DNS is needed (static IP or pinned cert). Even if DNS is used, certificate pinning blocks spoofing.

For the Server to Cloud link, full TLS with CA-validated certificates and revocation checks are used.

So even if DNS is compromised or the network is hostile, MitM fails due to certificate validation.

10.4 Mutual TLS with TPM-Bound Certificates

The system implements mutual TLS (mTLS) where the IoT device has a device cert (TPM-bound) and the server authenticates the device. This prevents rogue device injection.

During device provisioning in the one-time or manufacturing phase, the device generates a private key inside TPM where the key never leaves TPM. A device certificate is issued with public key only. The certificate is bound to Device ID and TPM key. The server stores device certificate and device identity metadata. This results in strong, hardware-rooted identity.

During runtime communication in the mTLS handshake, the client sends ClientHello, Certificate (Device cert), and CertVerify (TPM proves key ownership). The server authenticates the device. Then the server sends ServerHello and Certificate (Server cert). The device authenticates server. Both sides verify identity with no shared secrets, no spoofing, and no rogue device injection possible.

10.5 Why TPM-Bound Certificates Matter

Without TPM, an attacker can clone keys and fake devices are possible. With TPM, the private key is non-extractable and bound to hardware, making cloning cryptographically infeasible. Even if firmware is copied, the identity cannot be copied.

10.6 Secure Boot and Attestation

Secure Boot operates locally where the bootloader verifies firmware signature and chain of trust up to ROM. If firmware is modified, boot fails and the device never comes online.

Remote Attestation provides server-side verification. The device sends TPM quotes which include PCR values (hashes of boot components) signed by TPM attestation key. The server verifies TPM signature and compares PCR values with known-good firmware measurements, then accepts or rejects the device. The logic is: if valid cert and valid attestation, allow connection, else reject device.

10.7 Why Combining mTLS and Attestation is Powerful

For a rogue device threat, mTLS blocks it. For cloned firmware, both mTLS and attestation block it. For key extraction, mTLS blocks it. For firmware tampering, attestation blocks it. For MITM, mTLS blocks it. So identity and integrity are both covered.

An important design detail is that mTLS alone is not enough because a compromised device with valid cert could still connect. Attestation alone is not enough because a malicious but unregistered device could attest correctly. Together they provide Zero Trust.

The system enforces mutual authentication using TPM-bound device certificates (mTLS) to prevent unauthorized device access. Additionally, secure boot and remote attestation ensure that only devices running verified firmware are permitted to communicate with the server, effectively preventing firmware tampering and rogue device injection.

11 Key Technical Components

11.1 Physical Unclonable Functions (PUFs)

PUF-derived identities are stabilized using fuzzy extractors to tolerate environmental variations without compromising security. PUFs produce outputs based on manufacturing variations unique to each chip. However, noise, temperature, and aging can cause slight variations in output.

A fuzzy extractor has two phases. During enrollment, it takes a noisy PUF output and generates a stable secret key and some helper data (non-secret), then stores the helper data (not the key). During reconstruction, it takes a new noisy PUF output, uses helper data, and recovers the same secret key. The attacker can see helper data but still cannot derive the key.

11.2 Trusted Platform Module (TPM)

The controller is treated as a Trusted Computing Base (TCB) and is protected using a hardware-backed secure enclave (TPM). Attacks requiring physical tampering of the controller are considered infeasible and out of scope. In the event of detected physical tampering, the controller transitions to a fail-secure state and disables further network participation.

The TPM cannot be bypassed because it is in a secure enclave. It's not easily bypassable. In order to bypass the TPM, the attacker has to physically tamper with the controller. That is not feasible in any sense. If the controller is physically compromised only, then the man-in-the-middle attack may occur, but that is way too infeasible.

The system declares the controller as a trustable computing base because the TPM and other enclaves will ensure that it will not get tampered. Even if an attacker tries to physically compromise, a plan can be set that will prevent the device from even hooking up further. Physical hardware-level features can be implemented that will prevent the device from properly establishing the connection if a physical compromise of the controller occurs.

11.3 Device Enrollment

The enrollment is very strong for the devices to be connected with a microcontroller. The controller has some specific features, and using those specific features only can the devices be connected to the controller. Those specific features will be provided by the manufacturer itself, and using that particular special feature only can any device connect to the microcontroller.

This means the system is constraining devices to a particular manufacturer. That's a drawback and it's acknowledged. But it will enhance the security very, very much. Device authentication relies on manufacturer-provisioned hardware identities, restricting enrollment to authorized devices. While this limits heterogeneity, it significantly strengthens system integrity and prevents unauthorized device introduction.

There's also a possibility of administrator-mediated approval, and this will be looked into if there is a really highly important safety-critical application.

11.4 Replay and Relay Protection

Even with PUFs, an attacker can record messages and replay authentication transcripts. The server will actually initiate a communication medium wherein the two controllers can communicate with one another for a certain amount of time using encryptions that are known to them only. Using nonces and other techniques, replay or relay attacks cannot be done by the controller.

Controllers communicate using encryption known only to them, with nonces. Nonces must be fresh, unpredictable, and bound to session context.

12 Threat Model and Security Analysis

12.1 Formal Threat Assumptions

The local network is assumed to be untrusted and potentially malicious. Subsystem controllers are treated as trust anchors and form the Trusted Computing Base. Physical compromise of a controller is considered out of scope and results in localized system failure.

12.2 Controller as High-Value Target

If an attacker compromises one controller, all devices under it are compromised and the attacker gets trusted path to server. This is not a flaw but a known tradeoff. The mitigation includes secure boot on controller, controller identity pinned at server, periodic controller re-authentication, and controller attestation at startup.

Controllers are treated as trusted computing bases and are protected using secure boot and mutual authentication with the central server.

12.3 Device Authentication Requirements

Device authentication must not depend on the network alone. Basing it only on MAC address, IP behavior, or timing is bad. The good approach is that the device proves

cryptographic identity using pre-provisioned key, certificate, or secret, and challenge-response authentication.

Otherwise an attacker can impersonate devices and ARP spoofing becomes secondary. If the controller accepts traffic before authentication completes, then an attacker can race the legitimate device. The required rule is that unauthenticated devices must be in a "quarantine" state with no packet forwarding until authentication succeeds. This is crucial and strengthens the design immensely.

12.4 Controller Impersonation Prevention

A controller cannot authenticate or admit devices unless it has first authenticated itself to the server. This ensures no rogue controllers, no shadow controllers, and no early boot abuse.

Controllers operate in a restricted pre-authentication mode and cannot admit or forward device traffic until successful mutual authentication with the central server.

12.5 Security Coverage Summary

Threat	**Status**
ARP poisoning	Neutralized (not relied upon)
Boot-time MitM	Prevented
Rogue device	Prevented
Device impersonation	Prevented
Replay / relay	Prevented
Controller compromise	Out of scope, fail-secure
Scalability	Good
Offline operation	Fully supported

Table 1: Threat Coverage Analysis

13 System Design Principles

13.1 Zero Trust Architecture

The system follows zero trust principles where the LAN is assumed hostile, trust is not derived from addresses, and the system relies on hardware-backed identity. Many projects fail because they stop at "we'll detect ARP spoofing", but this approach goes two layers deeper.

13.2 Defense in Depth

The system implements multiple layers of defense:

1. Hardware-rooted identity (PUF/TPM)
2. Mutual authentication (mTLS)

3. Secure boot and attestation
4. Quarantine before authentication
5. Watchlist and baiting mechanisms
6. Controller-based admission control

13.3 Addressing the Bootstrapping Problem

The bootstrapping problem asks how a brand-new device gets bound to a controller securely. If enrollment is weak, an attacker can enroll itself first or race the legitimate device.

The system uses manufacturer-provisioned trust where the device certificate is signed by CA and the controller trusts that CA. This is the chosen approach for the current implementation.

14 Conclusion

The proposed system provides a comprehensive approach to preventing man-in-the-middle attacks in offline embedded systems through a carefully designed hierarchical trust architecture. By treating controllers as trust anchors with hardware-backed security, implementing mutual authentication with TPM-bound certificates, and enforcing strict admission control with quarantine-before-authentication, the system effectively neutralizes common attack vectors including ARP poisoning, DNS spoofing, and session hijacking.

The system design eliminates the attacker's ability to be on-path or in control of critical communication channels. Even when extended to online operation, the intermediate trusted server architecture maintains strong security guarantees while validating cloud services using standard public key infrastructure.

The acknowledgment of trade-offs, such as manufacturer-bound device constraints and controller compromise scenarios, demonstrates a mature understanding of security engineering. The explicit definition of threat boundaries and trust assumptions makes the architecture defensible against technical scrutiny.

Future work will focus on implementation details, performance evaluation, and validation of the security mechanisms in realistic embedded environments with resource-constrained devices.