Putting the S in IoT

Building Secure Connections with ESP-IDF

A Security-First Approach to ESP-IDF Networking

Euripedes Rocha

Espressif Brasil Summit 2025

Agenda

1. Presentation Goal: Security layer by layer
2. Cryptography Fundamentals
    1. Cryptographic Security Objectives
    2. Hybrid Cryptography
    3. Generic Secure Protocol Handshake
3. Channel Security Foundation
    1. Common WiFi Security Threats
    2. ESP-IDF WiFi Security Support
    3. Maximum Security Configuration
    4. Practical Configuration
4. Configuration Comparison
5. Certificate Validation
    1. Certificate-Based Trust
    2. Certificate Bundle
    3. Global CA Store
6. DNS Security
    1. DNS Security Problem
    2. ESP-DNS Component
7. MQTT Secure Connection Solution
    1. MQTT Security Challenge
    2. Plain TCP MQTT
    3. Secure MQTT Transport
    4. Broker Verification Options
    5. Username/Password Authentication
    6. Certificate-Based Authentication
    7. Digital Signature Authentication
    8. Complete Secure MQTT Solution
8. Hardware Security
    1. Device Identity Challenge
    2. eFuse Root of Trust
    3. ESP32 Hardware Security Reference
    4. Generate Keys & Certificate
    5. Configure DS Peripheral
    6. Runtime MQTT Integration
9. Complete ESP-IDF Security Solution
10. Thank You!

Presentation Goal: Security layer by layer

WiFi Channel Security
WPA3 encrypted wireless

Secure DNS Resolution
DNS-over-HTTPS/TLS

Broker Validation
TLS + certificate verification

Mutual Authentication
Client certificate auth

Hardware Identity
eFuse + Digital Signature

Cryptography Fundamentals

Cryptographic Security Objectives

Confidentiality

Protect data from unauthorized access

Integrity

Ensure data hasn’t been modified

Authentication

Verify identity of communicating parties

Non-repudiation

Prevent denial of actions performed

Hybrid Cryptography

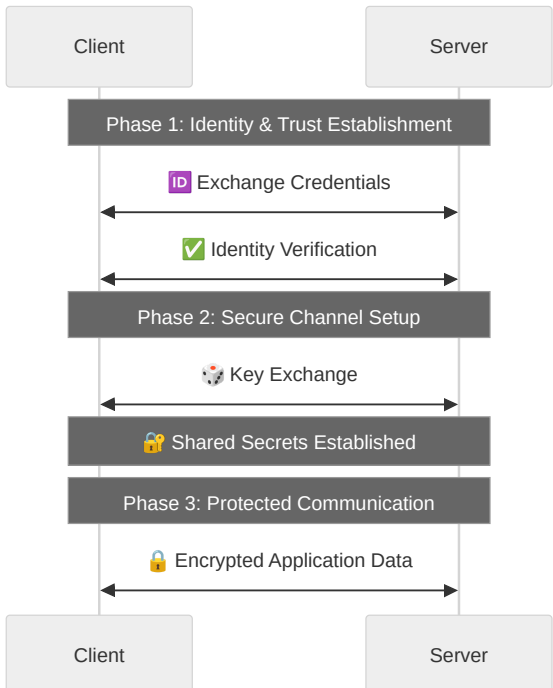
Asymmetric Cryptography

- **Key Pairs**: Public + Private keys
- **Usage**: Authentication, key exchange
- **Examples**: RSA, ECDSA, DH

Symmetric Cryptography

- **Shared Secret**: Same key encrypt/decrypt
- **Usage**: Bulk data encryption
- **Examples**: AES, ChaCha20

Generic Secure Protocol Handshake



Channel Security Foundation

Common WiFi Security Threats

🚨 Attack Vectors

WEP Vulnerabilities

- Weak 40/104-bit encryption
- IV collision attacks
- Crackable in minutes

WPA2 Dictionary Attacks

- Offline password cracking
- 4-way handshake capture
- No forward secrecy

🛡️ Management Frame Attacks

Deauth/Disassoc Spoofing

- Force client disconnection
- Man-in-the-middle setup
- Network disruption (DoS)

Evil Twin APs

- Rogue access points
- Credential harvesting
- Traffic interception

ESP-IDF WiFi Security Support

🔐 WPA3-Personal

- SAE Authentication
- Dictionary Attack Resistant
- Forward Secrecy
- H2E & Hunt-and-Peck

🏢 WiFi Enterprise

- WPA2/WPA3-Enterprise
- EAP-TLS, PEAP, TTLS
- EAP-FAST Support
- RADIUS Integration

🛡️ Security Features

- Protected Management Frames
- Enhanced Open (OWE)
- Certificate Validation
- Secure Provisioning

🔧 ESP-IDF Configuration Support

Menuconfig Options • Runtime API • Certificate Management • Event Handling

Maximum Security Configuration

```
wifi_config_t secure_config = {
    .sta = {
        .ssid = "SecureNetwork",
        .password = "StrongPassword123",
        .threshold.authmode = WIFI_AUTH_WPA3_PSK,        // Force WPA3
        .pmf_cfg = {
            .required = true                              // Override default (Optional)
        },
        .sae_pwe_h2e = WPA3_SAE_PWE_HASH_TO_ELEMENT,    // Override default (Both)
        .transition_disable = true,                       // Override default (false)
    }
};
```

⚡ ESP-IDF Defaults vs Override

```
# WPA3 enabled by default in ESP-IDF
CONFIG_ESP_WIFI_ENABLE_WPA3_SAE=y      # ✅ Default: enabled
# PMF Optional by default - we override to Required
# Transition allowed by default - we override to disabled
```

Practical Configuration

```
wifi_config_t practical_config = {
    .sta = {
        .ssid = "MyNetwork",
        .password = "MyPassword123",
        .threshold.authmode = WIFI_AUTH_WPA2_PSK,        // WPA2 minimum
        .pmf_cfg = {
            .required = false                             // ✅ Use default (Optional)
        },
        .sae_pwe_h2e = WPA3_SAE_PWE_BOTH,                // ✅ Use default (Both)
        .transition_disable = false,                      // ✅ Use default (allowed)
    }
};
```

🔧 ESP-IDF Secure Defaults

```
CONFIG_ESP_WIFI_ENABLE_WPA3_SAE=y      # ✅ Default: WPA3 enabled
      # ✅ Default: PMF Optional (connects when supported)
    # ✅ Default: Transition allowed (backward compatibility)
        # ✅ Default: SAE PWE Both (H2E + Hunt-and-Peck)
```

Configuration Comparison

🛡️ Practical Approach

- WPA2 minimum, WPA3 preferred
- **PMF Optional** for wider compatibility
- **Transition allowed** for legacy networks
- **Works with most networks**

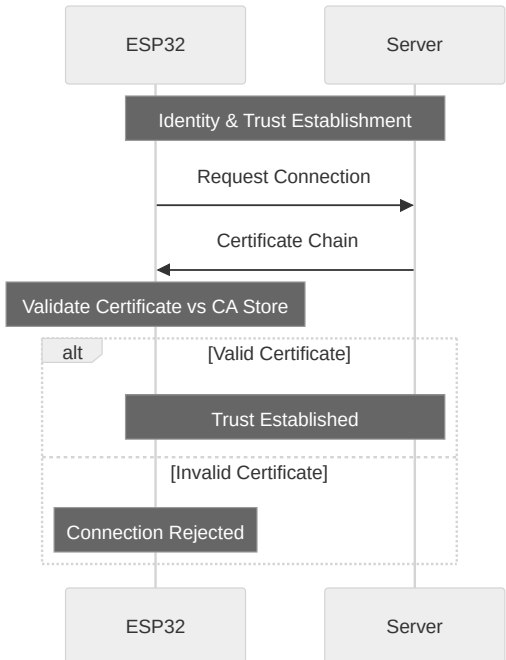
✅ ESP-IDF Smart Defaults

- **Auto WPA3 upgrade**: Connects WPA3 when available, falls back to WPA2
- **PMF when supported**: Enables PMF automatically if AP supports it
- **Side-channel protection**: H2E method available by default
- **Maximum compatibility**: Works with 99% of existing networks

Best Practice: Start with practical config, upgrade to maximum security in controlled environments

Certificate Validation

Certificate-Based Trust



Credential Exchange

- Server presents X.509 certificate
- Contains public key + identity claims
- Signed by trusted Certificate Authority

Certificate Bundle

📦 What is Certificate Bundle?

130+ root CA certificates (Mozilla Root Store)

- Same CAs used by web browsers
- Auto-updated with ESP-IDF releases
- ~60KB flash

Perfect for:

- ☁️ **Cloud services** (AWS IoT, Azure IoT)
- 🌐 **Public APIs** (GitHub, REST endpoints)
- 🔒 **Let's Encrypt** certificates

⚙️ Enable in menuconfig:

```
Component config → mbedTLS → Certificate Bundle
  → [*] Enable trusted root certificate bundle
```

Configuration:

```c
#include "esp_crt_bundle.h"

esp_tls_cfg_t cfg = {
    .crt_bundle_attach = esp_crt_bundle_attach,
    .skip_common_name = false,  // Verify hostname
};
```

Works with all protocols:

- HTTPS, MQTT, WebSockets, custom TLS

Global CA Store

Benefits:

- Runtime certificate loading
- Memory efficient for multiple connections
- Centralized management

Configuration:

```
esp_tls_set_global_ca_store(
    server_root_cert_pem_start,
    server_root_cert_pem_end - server_root_cert_pem_start
);

esp_tls_cfg_t cfg = {
    .use_global_ca_store = true,
};

esp_tls_free_global_ca_store();
```

Example: `examples/protocols/https_request/`

DNS Security

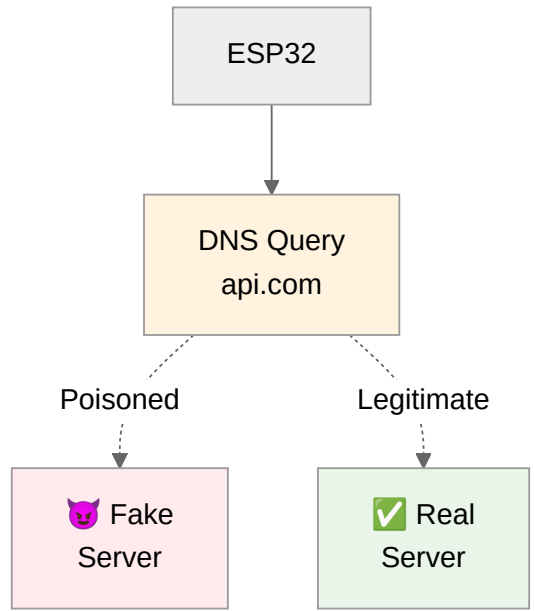
DNS Security Problem

DNS Vulnerabilities

- 🔓 **Unencrypted**: Queries in plaintext
- 🎯 **Cache poisoning**: Fake responses
- 👀 **Traffic analysis**: Monitoring behavior
- 🚫 **DNS hijacking**: Malicious redirects

Risk: ESP32 connects to wrong server

- ❌ No response verification
- ❌ Complete security bypass
- ❌ Firmware updates compromised



ESP-DNS Component

Protocol Support

- **UDP/TCP DNS (53)**: ❌ Plaintext, ⚡ Fast
- **DoT (853)**: ✅ TLS encrypted, memory overhead
- **DoH (443)**: ✅ HTTPS encrypted, memory overhead

🛡️ Security Features

- Certificate validation with ESP-IDF bundle
- Protection against DNS spoofing
- Encrypted query/response (DoT/DoH)

⚙️ Implementation

- Drop-in replacement for system DNS
- Standard `getaddrinfo()` interface
- Automatic fallback mechanisms
- Component: `esp-protocols/esp_dns`
- Add: `idf.py add-dependency "espressif/esp_dns^0.1.0"`

Example: `esp-protocols/components/esp_dns/examples/esp_dns_b`

MQTT Secure Connection Solution

MQTT Security Challenge

The Problem

Plain MQTT offers:

- Unencrypted transmission
- No broker authentication
- No client authentication
- No message integrity validation

Attack Scenarios

- Traffic interception and analysis
- Broker impersonation attacks
- Unauthorized message injection
- Device impersonation

Our Security Solution

1. **Transport Security**: TLS encryption
2. **Broker Verification**: Certificate validation
3. **Client Authentication**: Multiple options
4. **Hardware Integration**: ESP32 security features

ESP-IDF MQTT Examples

- `examples/protocols/mqtt/tcp/`
- `examples/protocols/mqtt/ssl/`
- `examples/protocols/mqtt/ssl_mutual_auth/`
- `examples/protocols/mqtt/ssl_ds/`

Plain TCP MQTT

ESPRESSIF

Basic MQTT Configuration

```
esp_mqtt_client_config_t mqtt_cfg = {
    .broker = {
        .address = {
            .uri = "mqtt://broker.hivemq.com:1883",
        }
    }
};
```

What This Provides

- Direct TCP connection to broker
- Unencrypted MQTT messaging
- Anonymous client connections
- Development and testing baseline

Security Limitations

- All messages transmitted in cleartext
- No server identity verification
- No client authentication required
- Vulnerable to MITM attacks

Use Cases:

- Development and testing
- Local network prototyping
- Non-sensitive IoT data

Example: `examples/protocols/mqtt/tcp/`

Secure MQTT Transport

TLS-Enabled MQTT

```
esp_mqtt_client_config_t mqtt_cfg = {
    .broker = {
        .address = {
            .uri = "mqtts://mqtt.eclipseprojects.io:8883",
        },
        .verification = {
            .crt_bundle_attach = esp_crt_bundle_attach,
        }
    }
};
```

Security Components

- **esp-tls**: TLS implementation layer
- **tcp_transport**: Secure socket transport
- **Certificate Bundle**: Pre-loaded root CAs

Security Improvements

- All MQTT traffic encrypted (TLS 1.2/1.3)
- Server identity verified via certificates
- Protection against eavesdropping
- Man-in-the-middle attack prevention

Example: `examples/protocols/mqtt/ssl/`

Broker Verification Options

Certificate-Based Verification

- Certificate Bundle - `.crt_bundle_attach`
- Custom Root CA - `.certificate`
- Global CA Store - `.use_global_ca_store`

Alternative Methods

- Pre-Shared Key - `.psk_hint_key`
- Custom Common Name - `.common_name`

Custom Root CA Example

```
extern const uint8_t ca_cert_pem_start[]
  asm("_binary_ca_cert_pem_start");

.verification = {
    .certificate = (const char *)ca_cert_pem_start,
}
```

Username/Password Authentication

Basic Configuration

```
.credentials = {
    .username = "device_12345",
    .client_id = "esp32_sensor_01",
    .authentication = {
        .password = "secure_device_password",
    }
}
```

All Credential Fields

- `.username` - MQTT username
- `.client_id` - MQTT client identifier
- `.authentication.password` - MQTT password

Advantages:

- Simple implementation
- Easy credential rotation
- Broker-managed users
- Standard MQTT feature

Best Practices:

- Store credentials in encrypted NVS
- Use strong, unique passwords
- Implement credential rotation
- Monitor authentication failures

Certificate-Based Authentication

Mutual TLS Configuration

```
extern const uint8_t client_cert_pem_start[]
  asm("_binary_client_crt_start");
extern const uint8_t client_key_pem_start[]
  asm("_binary_client_key_start");

.credentials = {
    .authentication = {
        .certificate = (const char *)client_cert_pem_start
        .certificate_len = 0, // PEM null-terminated
        .key = (const char *)client_key_pem_start,
        .key_len = 0, // PEM null-terminated
        .key_password = "optional_key_password",
        .key_password_len = 21,
    }
}
```

Security Benefits:

- Cryptographically strong authentication
- No password transmission
- Unique device identity
- Non-repudiation

Digital Signature Authentication

DS Peripheral Configuration

```
.credentials = {
    .authentication = {
        .ds_data = esp_secure_cert_get_ds_ctx(),
    }
}
```

Supported Devices

- ESP32-S2, ESP32-S3
- ESP32-C3, ESP32-C5, ESP32-C6, ESP32-C61
- ESP32-H2, ESP32-P4

Authentication Field

- `.ds_data` - DS context handle

Security Benefits:

- RSA private key never exposed in software
- Hardware-accelerated RSA signing
- Tamper-resistant authentication
- Uncompromisable device identity

ESP-IDF Reference:

`examples/protocols/mqtt/ssl_ds/`

Complete Secure MQTT Solution

Encrypted Transport

- TLS 1.2/1.3 with certificate bundle

Mutual Authentication

- Server verification via certificates
- Client authentication with DS peripheral

Production Ready

- Hardware-backed device identity
- Complete protection against MITM attacks

MQTT Security Stack:

- Transport encryption (TLS)
- Broker verification (certificates)
- Client authentication (hardware)

Hardware Security

Device Identity Challenge

ESPRESSIF

What We've Built So Far

✅ **Secure WiFi** (WPA3 encryption)

✅ **Secure DNS** (DoH/DoT resolution)

✅ **Certificate Validation** (CA bundle/custom)

✅ **MQTT Authentication** (certificates/passwords)

🤔 **But private keys still in software...**

Next Hardening Layer: Hardware Security

🔒 **Hardware-bound private keys**

- Never exposed to software

⚡ **Hardware-accelerated crypto**

- DS/HMAC peripherals

🏭 **Immutable device identity**

- eFuse root of trust

🛡️ **Tamper-resistant authentication**

- Uncompromisable credentials

eFuse Root of Trust

eFuse Architecture

🔒 One-Time Programmable Memory

- Hardware-protected storage

🔑 256-bit Key Blocks

- Up to 6 blocks on newer devices
- Purpose-bound usage
- Hardware protection

Root of Trust Capabilities

```c
// Burn key to eFuse (one-time)
uint8_t key[32];
esp_fill_random(key, sizeof(key));
esp_efuse_write_key(EFUSE_BLK_KEY0,
    ESP_EFUSE_KEY_PURPOSE_HMAC_DOWN_DIGISIGN,
    key, sizeof(key));

// Protect from software access
esp_efuse_set_read_protect(EFUSE_BLK_KEY0);
```

ESP32 Hardware Security Reference

Cryptographic Accelerators

Symmetric Cryptography

- **AES**: All devices

Flash Encryption

- **XTS-AES**: ESP32-S2, ESP32-S3, ESP32-C3, ESP32-C5, ESP32-C6, ESP32-P4
- **Custom Algorithm**: ESP32

Asymmetric Cryptography

- **RSA**: All devices
- **ECC**: ESP32-C5, ESP32-C6

Hash & Authentication

- **SHA**: All devices
- **HMAC**: ESP32-S2, ESP32-S3, ESP32-C3, ESP32-C5, ESP32-C6, ESP32-P4
- **Digital Signature (DS)**: ESP32-S2, ESP32-S3, ESP32-C3, ESP32-C5, ESP32-C6, ESP32-P4
- **ECDSA Peripheral**: ESP32-C5

Security Foundation

- **MPI (Large Number Operations)**: All devices
- **RNG**: All devices
- **eFuse Integration**: All devices

Generate Keys & Certificate

ESPRESSIF

Step 1: Generate RSA Private Key

```
# Generate 2048-bit RSA private key
openssl genrsa -out client.key 2048

# Verify key generation
openssl rsa -in client.key -text -noout
```

Step 2: Generate Certificate

```
# Generate Certificate Signing Request
openssl req -out client.csr -key client.key -new \
  -subj "/CN=ESP32-C5-Device/O=MyCompany"

# Option 1: Get certificate from CA
# (Submit CSR to Certificate Authority)

# Option 2: Self-signed certificate for development
openssl x509 -req -days 365 -in client.csr \
  -signkey client.key -out client.crt
```

Configure DS Peripheral

Step 1: Install esp_secure_cert Tool

```
# Install the official ESP-IDF tool
pip install esp-secure-cert-tool

# Verify installation
configure_esp_secure_cert.py --help
```

Step 2: Configure DS with Generated Keys

```
# Configure DS peripheral with key and certificate
configure_esp_secure_cert.py \
  -p /dev/ttyUSB0 \
  --device-cert client.crt \
  --private-key client.key \
  --target_chip esp32c5 \
  --configure_ds

# Result: esp_secure_cert.bin created automatically
# HMAC key burned to eFuse
# DS parameters encrypted and stored
```

Runtime MQTT Integration

Step 1: Load DS Context & Certificate

```
// Load DS context from esp_secure_cert partition
esp_ds_data_ctx_t *ds_data = esp_secure_cert_get_ds_ctx();
if (ds_data == NULL) {
    ESP_LOGE(TAG, "Error reading DS data");
    return;
}

// Load device certificate
char *device_cert = NULL;
uint32_t len;
esp_err_t ret = esp_secure_cert_get_device_cert(&device_ce
if (ret != ESP_OK) {
    ESP_LOGE(TAG, "Failed to get device certificate");
    return;
}
```

Step 2: Configure MQTT Client

```
// Configure MQTT with hardware-backed authentication
const esp_mqtt_client_config_t mqtt_cfg = {
    .broker = {
        .address.uri = "mqtts://test.mosquitto.org:8884",
        .verification.certificate = server_cert_pem,
    },
    .credentials = {
        .authentication = {
            .certificate = device_cert,
            .key = NULL,         // No software key needed
            .ds_data = ds_data  // Hardware handles signi
        },
    },
};
```

🛡️ Complete ESP-IDF Security Solution

🔧 Implementation Journey

Problem Statement

- Complete secure MQTT architecture
- Defense-in-depth strategy

Channel Security

- WPA3-Personal configuration
- Secure WiFi provisioning

Certificate Validation

- TLS trust establishment
- ESP-IDF certificate bundle

🔒 Advanced Security

DNS Security

MQTT Security

- Hardware-backed authentication
- TLS client certificates

Hardware Security

- Digital Signature peripheral
- Hardware identity generation

Thank You!

Questions?

END