

Secure System Design

elaborato finale

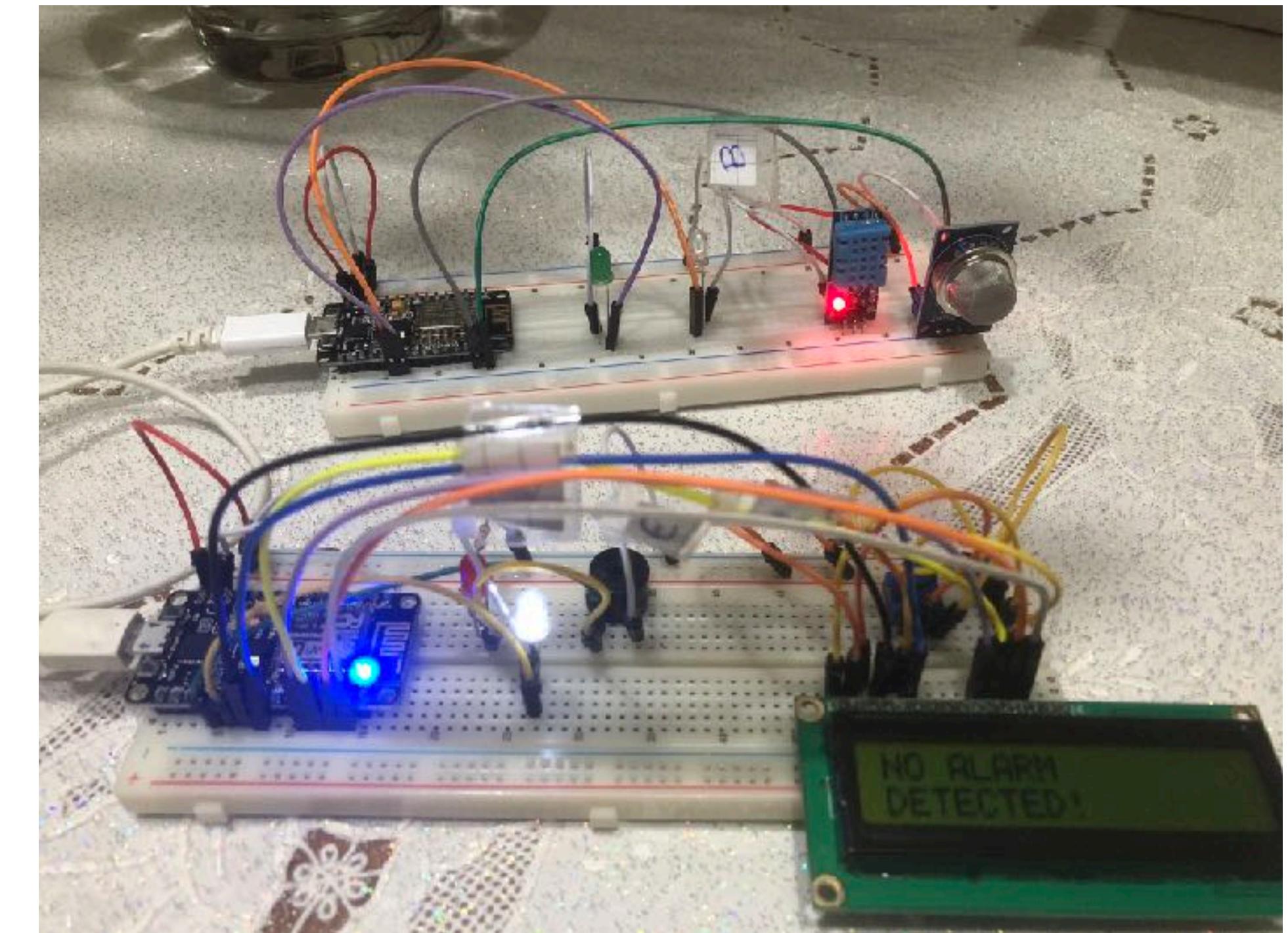
A cura di **Pasquale Tremante**
Marzo 2018

Il Progetto

Un sistema di sensoristica il cui compito è quello di monitorare l'ambiente per rilevare eventuali fughe di fumi e gas.

Il sistema è costituito da una coppia di nodi hardware, secondo un approccio Client-Server, dove il nodo **client** ha il compito di monitorare l'ambiente per captare presenze di gas e, in tal caso, avviare una comunicazione con il nodo **server** per dare l'allarme; quando quest'ultimo riceve il segnale di allerta, elabora l'emergenza (ad esempio contattando autorità competenti) ed invia una risposta al nodo client per segnalargli che l'allarme è stato ricevuto e gestito correttamente.

Entrambi i nodi sono connessi ad una medesima rete locale, all'interno della quale si scambiano i messaggi.

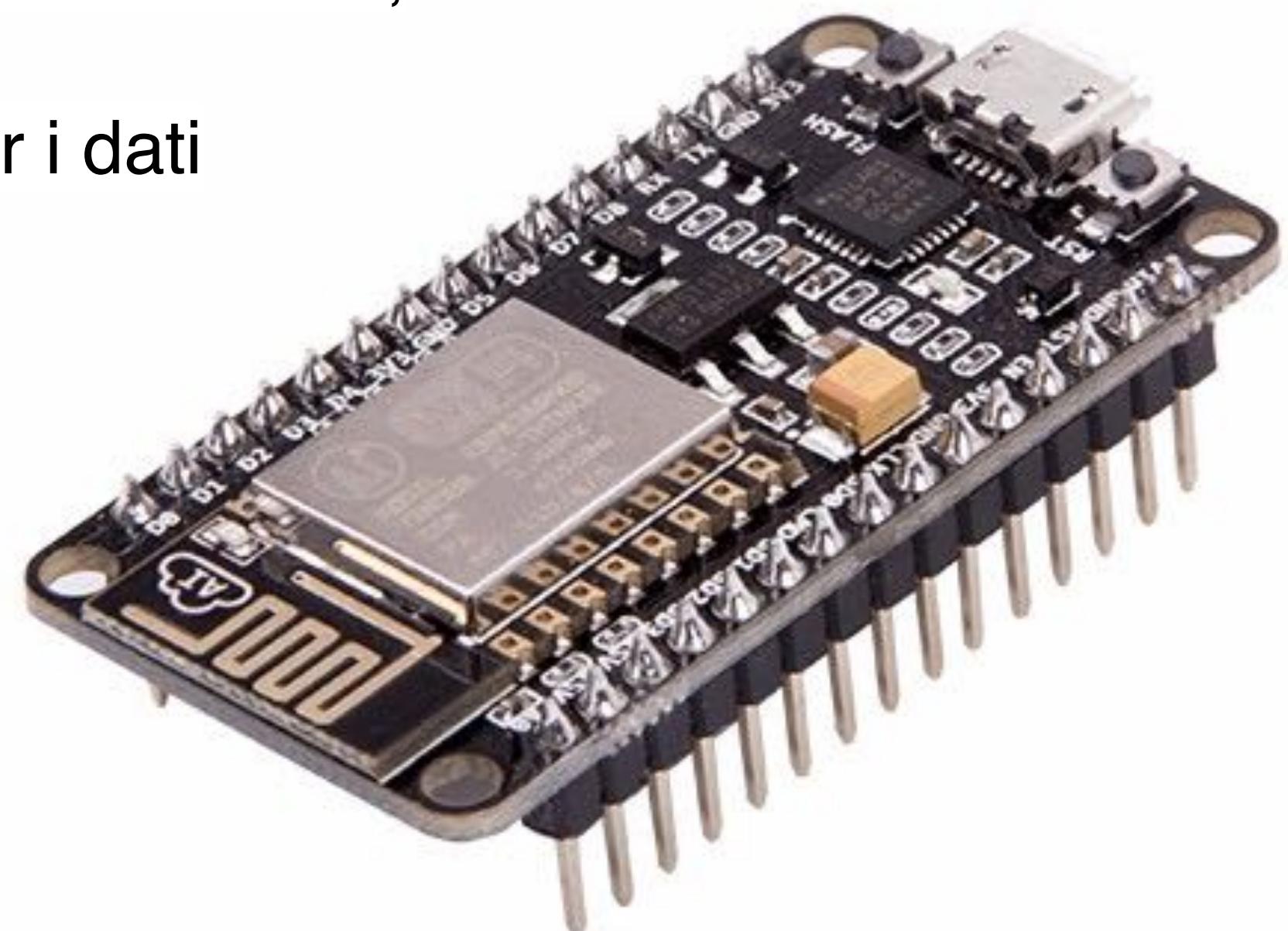


NodeMCU ed ESP8266

Lo sviluppo di questo progetto si basa sull'utilizzo di una piattaforma open source sviluppata specificatamente per l'IoT, chiamata NodeMCU.

La peculiarità di questa board è quella di montare il modulo Wi-Fi ESP8266 che comprende:

- una CPU Tensilica Xtensa LX106 che gira a 80MHz (su alcuni modelli può arrivare a 160MHz)
- una memoria flash esterna che varia da 512Mb a 4Mb a seconda del modello, in cui viene memorizzato il programma
- 64Kb di memoria RAM per le istruzioni e 96Kb di memoria RAM per i dati
- 512 bytes di EEPROM;
- Lo stack TCP/IP e la possibilità di collegarsi alle reti wifi b/g/n;
- 16 GPIO;
- UART / I2C / I2S/ SPI / 1 modulo ADC a 10bit.

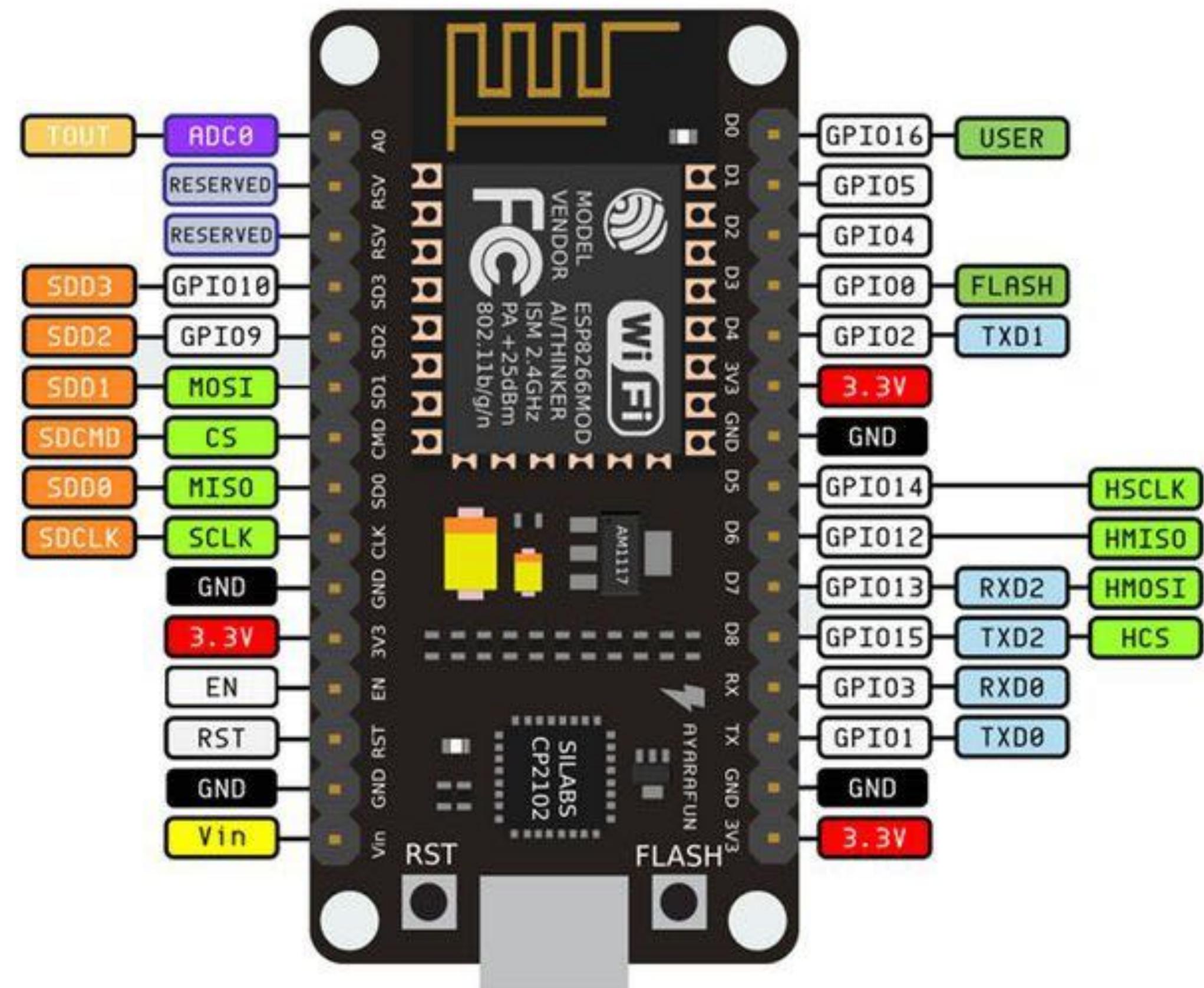


NodeMCU ed ESP8266

Il NodeMCU oltre a integrare il modulo ESP con tutte le funzionalità appena citate:

- Ha già installato il firmware NodeMCU al posto dell'interprete comandi AT di serie;
- Ha integrato un convertitore USB/seriale TTL;
- Ha integrato il regolatore di tensione a 3.3V;
- Sono presenti due pulsanti (reset e flash, quest'ultimo serve per riprogrammarne il firmware, cosa che sui moduli ESP si ottiene portando a massa il pin GPIO0).

Per lo sviluppo di progetti con l'ESP8266, è possibile utilizzare le librerie open-source (con relativa documentazione) disponibili su GitHub all'indirizzo: <https://github.com/esp8266/Arduino>



Cosa ho utilizzato?

Per il nodo client:

- 1 NodeMCU
- 1 Sensore MQ2 per rilevare gas combustibile e fumo;
- 1 Sensore DHT11 per rilevare temperatura e umidità;
- 1 led bianco per segnalare l'avvenuta connessione alla rete;
- 1 led verde per segnalare l'avvenuta ricezione dell'allarme da parte del nodo server.

Per il nodo server:

- 1 NodeMCU
- 1 led bianco per segnalare l'avvenuta connessione alla rete;
- 1 led rosso per segnalare la ricezione dell'allarme da parte del nodo client + 1 buzzer per emettere suono
- 1 schermo LCD per mostrare i valori di temperatura e umidità ricevuti dal client

Security

Affinché la comunicazione tra client e server avvenga in modo sicuro, i due nodi implementano il protocollo SSL.

Per realizzarlo è stato necessario usare:

- Generare le chiavi e i certificati digitali con OpenSSL;
- Applicare le funzioni offerte dalla libreria dell'ESP8266.

Generazione dei certificati digitali tramite OpenSSL

L'autenticazione SSL all'interno di questo progetto è **unilaterale**, ovvero, è il solo server ad autenticarsi presso il client. A tale scopo quindi è stato necessario generare un certificato digitale e una chiave privata RSA per il nodo server; ma prima ancora è stato generato un certificato digitale autofirmato per simulare un CA fittizia che validasse il certificato del server. Per fare questo abbiamo utilizzato i seguenti comandi con OpenSSL...

Generazione certificato digitale autofirmato CA

```
# 1024 or 512. 512 saves memory...
BITS=512
C=$PWD
pushd /tmp

#CREATION CA ROOT

#1) Generation of a RSA Private Key with keysize 512 bit and encrypted with -des3
openssl genrsa -out PasqualeCA_key.pem $BITS -des3 -passout dattebayo

#2) Geration of a self-signed certificate
openssl req -x509 -key PasqualeCA_key.pem -new -days 365 -out PasqualeCA_x509cert.pem -sha256

#3) Conversion of certificate in .der (to store on EEPROM of Arduino)
openssl x509 -in PasqualeCA_x509cert.pem -outform DER -out PasqualeCA_x509cert.cer
```

Generazione certificato digitale autofirmato CA

```
Tremantes-MBP:Desktop pastre23$ openssl genrsa -out PasqualeCA_key.pem 512 -des3 -passout dattebayo
Generating RSA private key, 512 bit long modulus
..+++++
.....+++++
e is 65537 (0x10001)
Tremantes-MBP:Desktop pastre23$ openssl req -x509 -key PasqualeCA_key.pem -new -days 365 -out PasqualeCA_x509cert.pem -sha256
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) []:IT
State or Province Name (full name) []:Italy
Locality Name (eg, city) []:Naples
Organization Name (eg, company) []:PasqualeCorp
Organizational Unit Name (eg, section) []:TeamBallo
Common Name (eg, fully qualified host name) []:127.0.0.1
Email Address []:pasquale.tremante@hotmail.it
Tremantes-MBP:Desktop pastre23$ openssl x509 -in PasqualeCA_x509cert.pem -outform DER -out PasqualeCA_x509cert.cer
Tremantes-MBP:Desktop pastre23$
```

Generazione chiave e certificato digitale per nodo Server

```
#Creation Server Certificate
cat > certs.conf <<EOF
[ req ]
distinguished_name = req_distinguished_name
prompt = no

[ req_distinguished_name ]
O = ServerNodeMCU
CN = 172.20.10.3
EOF

#1) Generation of a RSA Private Key with keysize 512 bit and encrypted with -des3
openssl genrsa -out ServerNodeMCU_key.pem $BITS -des3 -passout hokage

#2) Conversion of RSA key in .der (to store on EEPROM of Arduino)
openssl rsa -in ServerNodeMCU_key.pem -out ServerNodeMCU_key -outform DER

#3) Request for a digital certificate
openssl req -out ServerNodeMCU_x509cert.req -key ServerNodeMCU_key.pem -new -config certs.conf

#4) Request to CA to sign the request and generate a digital certificate for Server
openssl x509 -req -in ServerNodeMCU_x509cert.req -out ServerNodeMCU_x509cert.pem -sha256 -CAcreateserial
-days 5000 -CA PasqualeCA_x509cert.pem -CAkey PasqualeCA_key.pem

#5) Conversion of certificate in .der (to store on EEPROM of Arduino)
openssl x509 -in ServerNodeMCU_x509cert.pem -outform DER -out ServerNodeMCU_x509cert.cer #conversion
```

Indirizzo IP statico assegnato al nodo Server!

Distribuzione e conservazione dei certificati digitali e della chiave

Una volta che chiave e certificati sono stati generati, con il seguente comando li trasformiamo in un array di uint8 e li salviamo in file .h:

```
xxd -i ServerNodeMCU_key      | sed 's/.*{://' | sed 's/};//' | sed 's/unsigned.*//' > "$C/serverKey.h"
xxd -i ServerNodeMCU_x509cert.cer | sed 's/.*{://' | sed 's/};//' | sed 's/unsigned.*//' > "$C/serverCert.h"
xxd -i PasqualeCA_x509cert.cer   | sed 's/.*{://' | sed 's/};//' | sed 's/unsigned.*//' > "$C/caCert.h"
```

Fatto questo è possibile salvarli nella memoria flash dei nostri dispositivi, vediamo come...

Implementazione di SSL (lato server) (1/3)

```
#include <ESP8266WiFi.h>
#include <FS.h>
#include <time.h>
#include <LiquidCrystal.h>

const char* ssid      = "iphoneTremante";
const char* password = "12345678";

static const uint8_t x509[] PROGMEM = {
    #include "serverCert.h"
};

static const uint8_t rsakey[] PROGMEM = {
    #include "serverKey.h"
};

int pin_alarm = D8;
int pin_wifi_connected = D7;

// initialize the library by associating any needed LCD interface pin
// with the arduino pin number it is connected to
const int rs = D6, en = D5, d4 = D4, d5 = D3, d6 = D2, d7 = D1;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

// Create an instance of the server
// specify the port to listen on as an argument
WiFiServerSecure server(80);
```

Ehm...! Ne riparliamo
dopo! 😅

Salvataggio chiave privata e certificato del
server nella memoria.

Aggiungendo questa keyword, salviamo le
variabili all'interno della memoria flash del
nostro dispositivo, anziché nella SRAM.

- Necessary to implement TLS
- Necessary for connection and communication
- Other information

Implementazione di SSL (lato server) (2/3)

```
void setup()
{
    Serial.begin(115200);
    delay(10);

    //Connecting Wifi
    connectToWiFi();
    digitalWrite(pin_wifi_connected,HIGH);

    //Set certificate and private key
    server.setServerKeyAndCert_P(rsakey, sizeof(rsakey),
                                x509, sizeof(x509));

    // Start the server
    server.begin();
    Serial.println("Server started");

    // Print the IP address
    Serial.println(WiFi.localIP());

    // Synchronize time using SNTP.
    SynchronizeSNTP();

    //4) Initializing Pin
    pinMode(pin_alarm,OUTPUT);
    pinMode(pin_wifi_connected,OUTPUT);

    // set up the LCD's number of columns and rows:
    lcd.begin(16, 2);
}
```

```
void connectToWiFi()
{
    // Connect to WiFi network
    Serial.println();
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");
    Serial.println("WiFi connected");
}
```

Not necessary for
server

```
void SynchronizeSNTP() {
    // Synchronize time using SNTP. This is necessary to verify that
    // the TLS certificates offered by the server are currently valid.
    Serial.print("Setting time using SNTP");
    configTime(8 * 3600, 0, "pool.ntp.org", "time.nist.gov");
    time_t now = time(nullptr);
    while (now < 8 * 3600 * 2) {
        delay(500);
        Serial.print(".");
        now = time(nullptr);
    }
    Serial.println("");
    struct tm timeinfo;
    gmtime_r(&now, &timeinfo);
    Serial.print("Current time: ");
    Serial.print(asctime(&timeinfo));
}
```

Implementazione di SSL (lato server) (3/3)

```
void loop() {
    lcd.setCursor(0, 0);
    lcd.print("NO ALARM");
    lcd.setCursor(0, 1);
    lcd.print("DETECTED!");

    // Check if a client has connected
    WiFiClientSecure client = server.available();
    if (!client) {
        return;
    }

    // Wait until the client sends some data
    Serial.println("new client");
    unsigned long timeout = millis() + 3000;
    while(!client.available() && millis() < timeout){
        delay(1);
    }
    if (millis() > timeout) {
        Serial.println("timeout");
        client.flush();
        client.stop();
        return;
    }

    // Read and Print the request from CLIENT
    String req = client.readStringUntil('\r');
    Serial.println(req);
    Serial.println();
    client.flush();

    //Building response
    String response = "RESPONSE: ";
    // Send the response to the client
    if (req.indexOf("GAS_ALARM") != -1) {

        response += "GAS_ALARM_DETECTED\n";
        String temperature = "Temp: " + readUntil(req.substring(25), "\n");
        String humidity = "Humidity: " + readUntil(req.substring(38), "\n");
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print(temperature);
        lcd.setCursor(0, 1);
        lcd.print(humidity);
        send_alarm();

    } else {
        response += "INVALID_REQUEST\n";
    }

    String host = IpAddress2String(WiFi.localIP());
    response += "Host: " + host + "\n\r";
    client.print(response); //send response

    delay(1);
    Serial.println("Client disconnected");
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("NO ALARM");
    lcd.setCursor(0, 1);
    lcd.print("DETECTED!");
}
```

Implementazione di SSL (lato client) (1/2)

```
#include <dht11.h>
#include <time.h>
#include <ESP8266WiFi.h>
#include <WiFiClientSecure.h>

const char* ssid      = "iphoneTremante";
const char* password = "12345678";

const char* host = "172.20.10.3";
const int httpsPort = 80;

static const uint8_t caCert[] PROGMEM = {
    #include "caCert.h"
};

//Variables
int alarm_received = D6;
int pin_wifi_connected = D7;
int gas_alarm = D0;
int dht11_pin = D1;
DHT dht11 DHT;

//Function Setup
void setup() {
    //0) Initializing Serial
    Serial.begin(115200);
    delay(10);

    //4) Initializing Pin
    pinMode(alarm_received, OUTPUT);
    pinMode(pin_wifi_connected, OUTPUT);
    pinMode(gas_alarm, INPUT);
    pinMode(dht11_pin, OUTPUT);

    read_dht11(); //fictitious reading
}
```

Indirizzo IP assegnato staticamente al nodo server

Salvataggio nella memoria flash del certificato della CA che il client utilizzerà per verificare l'identità del server

In questa fase di definizione delle variabili e setup, non ci sono azioni particolarmente rilevanti da mostrare ai fini dell'implementazione di SSL; la connessione a internet e al server e la verifica del certificato viene fatta all'interno della funzione loop per evitare che il dispositivo rimanga inutilmente connesso alla rete.

Implementazione di SSL (lato client) (2/2)

```
void loop() {
    if(digitalRead(gas_alarm)==LOW) {

        //Connecting Wifi
        connectToWiFi();
        digitalWrite(pin_wifi_connected,HIGH); //turn on the led

        // Synchronize time using SNTP. This is necessary to verify that
        // the TLS certificates offered by the server are currently valid.
        SynchronizeSNTP();

        Serial.println(WiFi.localIP()); //Print the IP address

        //Create client to begin a connection
        WiFiClientSecure client;

        //Load root certificate in DER format into WiFiClientSecure object
        bool res = client.setCACert_P(caCert, sizeof(caCert));
        if (!res) {
            Serial.println("Failed to load root CA certificate!");
            while (true)
                yield();
        } else
            Serial.println("Success to load root CA certificate!");

        Serial.println("Client Ready!!");

        // Connect to remote server
        Serial.print("connecting to ");
        Serial.println(host);
        if (!client.connect(host, httpsPort)) {
            Serial.println("connection failed");
            WiFi.disconnect(); //Disconnect WiFi
            digitalWrite(pin_wifi_connected,LOW);
            return;
        } else
            Serial.println("Connected to server!");

        // Verify validity of server's certificate
        if (client.verifyCertChain(host)) {
            Serial.println("Server certificate verified");
        } else {
            WiFi.disconnect(); //Disconnect WiFi
            digitalWrite(pin_wifi_connected,LOW);
            Serial.println("ERROR: certificate verification failed!");
            return;
        }

        //Read data from temperature sensor
        read_dht10();

        //prepare message
        String request = "REQUEST: GAS_ALARM\n";
        request += "Temp: " + String(DHT.temperature) + "\n";
        request += "Humidity: " + String(DHT.humidity) + "\n";
        request += "Host: " + (String)host + "\n";
        request += "User-Agent: ESP8266\n\r";

        //send message
        client.print(request);
        Serial.println("request sent");

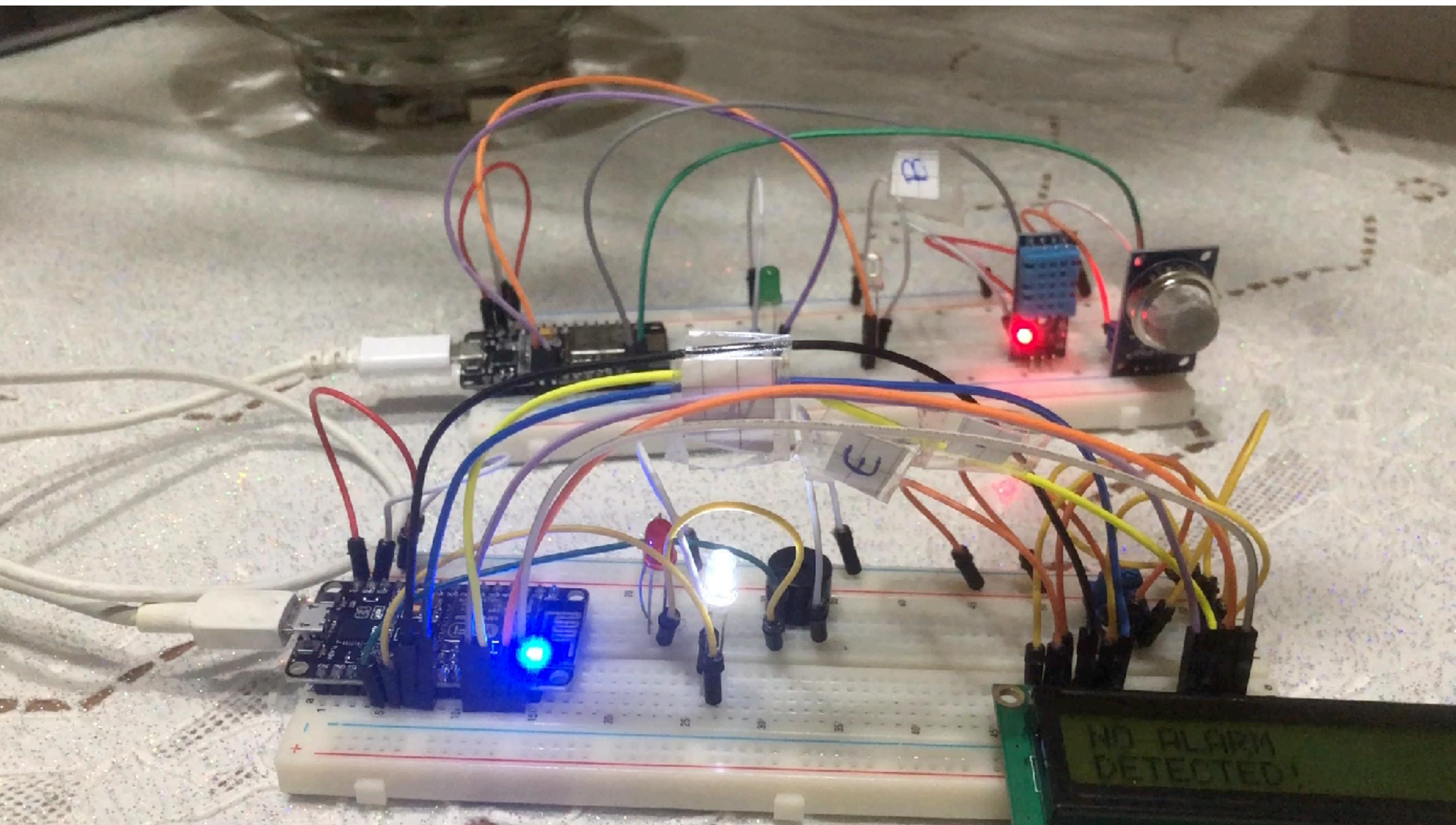
        while (client.connected()) {
            String response = client.readStringUntil('\r');
            Serial.println(response);
            if (response.indexOf("GAS_ALARM_DETECTED") != -1) {
                Serial.println("Alarm received");
                //Turn on green led
                digitalWrite(alarm_received,HIGH);
                delay(3000);
                digitalWrite(alarm_received,LOW);
            }
            break;
        }

        //Disconnect WiFi
        WiFi.disconnect();
        digitalWrite(pin_wifi_connected,LOW);
    }
    else //digitalRead(gas_alarm)==HIGH
        Serial.println("No Gas alarm detected!!");

    delay(1000);
}
```

Necessary for client

Live Video



Vulnerabilità

- **Autenticazione solo lato server:**

In realtà è solo un punto in sospeso, perché la libreria dell'ESP8266 offre anche dei metodi per l'autenticazione; inoltre, in alternativa, potremmo implementare una comunicazione a doppio canale TLS (quindi creare un certificato anche per il client validato dalla stessa CA del server).

- **Conservazione della chiave privata e Salvataggio password Wi-Fi:**

Purtroppo c'è poco da fare! Salvare tutto nella memoria flash attualmente è il male minore in quanto il sistema, a meno che i dispositivi fisici non finiscano nelle mani di un utente esperto, resta sicuro. Se i dati venissero salvati nella EEPROM del dispositivo stesso sarebbero addirittura alla merce di un utente assolutamente inesperto.

Conclusioni

Si conclude qui questo progetto che aveva lo scopo di mostrare come realizzare, in modo molto banale, una comunicazione sicura tra due nodi hardware connessi ad una stessa rete locale.

Abbiamo visto come il sistema risulti insicuro dal punto di vista dello storage delle informazioni, ma d'altronde da una board il cui valore di mercato oscilla tra i 3-10€...cosa possiamo pretendere?

Resta da mettere in chiaro comunque che l'implementazione dei metodi nei 2 sketch precedenti, andrebbe fatta in maniera più accurata; io mi sono solo limitato solo a mostrare in modo semplice come utilizzare le funzioni.

Mi auguro di essere stato chiaro e che il progetto sia stato di vostro chiarimento.

Fine