SecureVoting: IoT Voting using Flask and HTTPS

Giovanni Rapa

Table of **contents**

01

Project Purpose

Idea description

The idea

In view of the need to create a secure voting system already presented in the data security exam, I thought of implementing an IoT device that could guarantee secure voting.

The implementation

This is implemented via an esp32 consisting of four buttons, each of which allows a candidate to be voted for. The authentication that enables voting is implemented by an RFID chip that checks and assigns accounts on the blockchain.

02

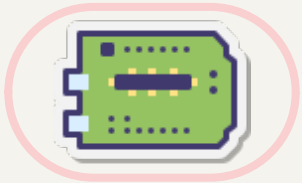
Technology used

Hardware, technology and libraries used

What I have **used**

Flask that uses SSL/TLS

The flask server operates an authentication system and implements encryption in order to make voting impossible to intercept.



ESP32 with RFID

The ESP32 initially authenticates the voter via RFID and then allows him/her to vote for one of the four proposed candidates. It also does not allow voting twice.

MySQL server and Ganache

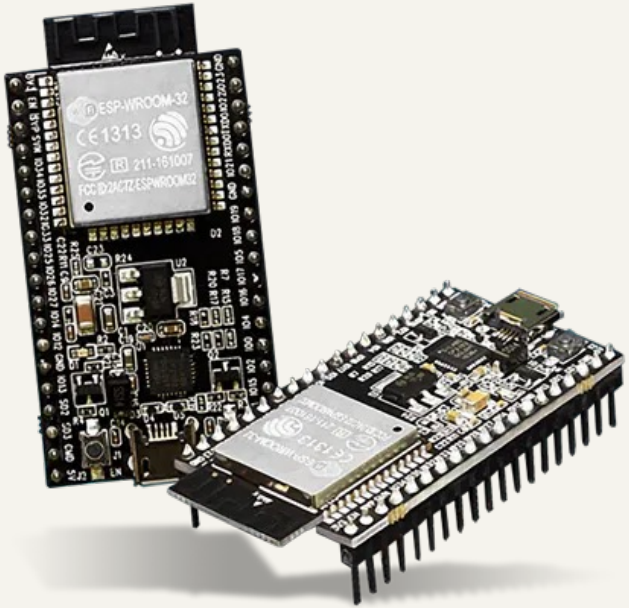
The MySQL server translates the RFID tag into blockchain accounts that are present on the Ganache blockchain. In addition, the server takes into account whether voting has already taken place or not.

Hardware

ESP32

The board **DevKitC v4** used for the development has 36 pins, **WiFi** and **Bluetooth** connectivity. It is based on chip **ESP 32-D0WDQ6** and has the following features:

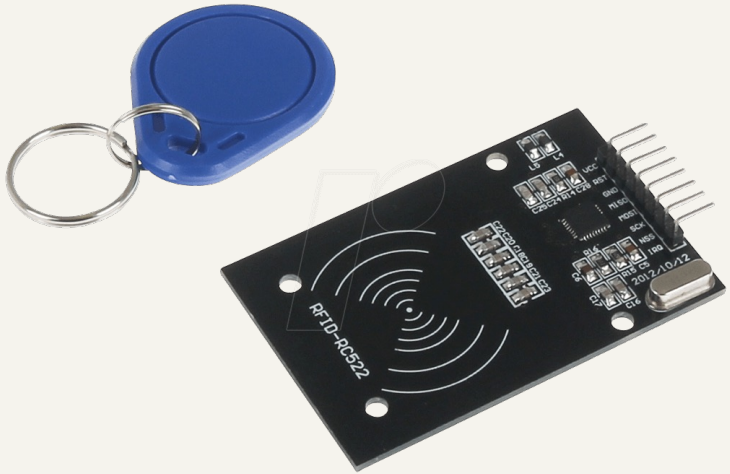
- Processor Xtensa 32-bit dual-core LX6
- 448 KBytes ROM
- 520 KBytes SRAM
- 4 MB EEPROOM
- WiFi 802.11 2.4GHz
- Bluetooth v4.2 BR/EDR and Bluetooth Low Energy (BLE)



RFID-RC522

The **RC522** is a **RFID module** that is based on the **MFRC522** controller from NXP semiconductors. The module can support **I2C**, **SPI and UART** communication. The **RC522** has the following features:

- 13.56MHz RFID module
- Operating voltage: 2.5V to 3.3V
- Communication : SPI, I2C protocol, UART
- Maximum Data Rate: 10Mbps
- Read Range: 5cm
- Current Consumption: 13-26mA



Software

Libraries used

On Flask:
- **Json**: to return errors with a specific error code
- **Web3**: for interacting with the blockchain
- **SSLify**: to implement encryption and force all requests into HTTPS

On ESP32:
- **WiFi**: provides access to WiFi functionality.
- **WiFiClientSecure**: provides secure communication over WiFi.
- **HTTPClient**: allows sending HTTP requests.
- **SPI**: is used for communication with peripherals using the Serial Peripheral Interface (SPI) bus.
- **MFRC522**: is a library for the MFRC522 NFC (Near Field Communication) reader.

03

Implementation

How everything was implemented

Arduino code

Loop function

```
void loop() {

  // Controls if the boolean variable State is True o False, it's implemented like a mutex
  if(state){
    authenticate();
  } else {
    vote(actualBlockchainID);
  }
}
```

Setup function

```
void setup() {

  // Setting up the Baud rate
  Serial.begin(9600);

  // Inizializing the WiFi Connection
  WiFi.begin(ssid, password);
  Serial.println("Connecting");
  while(WiFi.status() != WL_CONNECTED) {
    digitalWrite(RED_LED, HIGH);
    delay(500);
    Serial.print(".");
    digitalWrite(RED_LED, LOW);
  }
  Serial.println("");
  Serial.print("Connected to WiFi network with IP Address: ");

  // Turn the Green Led on success
  digitalWrite(GREEN_LED, HIGH);
  delay(LED_TIMING);
  digitalWrite(GREEN_LED, LOW);

  // Setting the CA Certificate for HTTPS, client.setInsecure()
  client.setCACert(root_ca);
  client.setInsecure();

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

  // Set the pins
  pinMode(FIRST_BUTTON, INPUT_PULLUP);
  pinMode(SECOND_BUTTON, INPUT_PULLUP);
  pinMode(THIRD_BUTTON, INPUT_PULLUP);
  pinMode(FOURTH_BUTTON, INPUT_PULLUP);
  pinMode(GREEN_LED, OUTPUT);
  pinMode(RED_LED, OUTPUT);
```

```
  // Set RFID
  SPI.begin();
  rfid.PCD_Init();

  // Controls if Ganache is connected with the Flask Webserver
  if(isConnectedGanache()){

    // Turns on Green Led on success
    Serial.println("ESP32 connected with Ganache");
    digitalWrite(GREEN_LED, HIGH);
    delay(LED_TIMING);
    digitalWrite(GREEN_LED, LOW);
  } else {

    // Turns on Red Led on failure and stops the program
    Serial.println("Error! Check Ganache connection!");
    digitalWrite(RED_LED, HIGH);
    delay(LED_TIMING);
    digitalWrite(RED_LED, LOW);
    exit(-1);
  }
}
```

Check Ganache

```
// This function controls if Ganache is connected to the Flask Web Server by returning a boolean
bool isConnectedGanache(){

  if(WiFi.status()== WL_CONNECTED){

      // There we forge the HTTPS GET
      HTTPClient https;
      String serverPath = serverName + "/isConnectedGanache";
      https.begin(client, serverPath.c_str());

      // We add this token called "Authorization" because in the back-end of the webserver only
      // request with this secret token will be accepted
      https.addHeader("Authorization", secureToken);

      // Getting the response code after executing a GET
      int httpsResponseCode = https.GET();

      // If OK then read the payload
      if (httpsResponseCode == 200) {
        String payload = https.getString();
        if(payload.equals("True")){
          return true;
        }
      }
      else {
        Serial.print("Error code: ");
        Serial.println(httpsResponseCode);
      }

      // Close HTTPS connection
      https.end();

    }
    else {
      Serial.println("WiFi Disconnected");
    }

    return false;
}
```

Vote Function

```
// This function uses the Blockchain ID took after the authentication to vote
void vote(String BlockchainID){

  // Setting the states of Buttons
  firstCurrentState = digitalRead(FIRST_BUTTON);
  secondCurrentState = digitalRead(SECOND_BUTTON);
  thirdCurrentState = digitalRead(THIRD_BUTTON);
  fourthCurrentState = digitalRead(FOURTH_BUTTON);

  // Checking if any of them is pressed
  if (firstLastState == HIGH && firstCurrentState == LOW)
    voteForPost(actualBlockchainID, 0);
  else if (secondLastState == HIGH && secondCurrentState == LOW)
    voteForPost(actualBlockchainID, 1);
  else if (thirdLastState == HIGH && thirdCurrentState == LOW)
    voteForPost(actualBlockchainID, 2);
  else if (fourthLastState == HIGH && fourthCurrentState == LOW)
    voteForPost(actualBlockchainID, 3);

  // Keeping the state of Buttons update
  firstLastState = firstCurrentState;
  secondLastState = secondCurrentState;
  thirdLastState = thirdCurrentState;
  fourthLastState = fourthCurrentState;
}
```

Vote POST request

```
// That's the real Vote function that does an HTTPS Post to the Flask Web Server
void voteForPost(String uid, int candidateID){
  if(WiFi.status()== WL_CONNECTED){

      // There we forge the HTTPS POST
      HTTPClient https;
      String serverPath = serverName + "/vote";
      https.begin(client, serverPath.c_str());
      https.addHeader("Content-Type", "application/x-www-form-urlencoded");

      // We add this token called "Authorization" because in the back-end of the webserver only
      // request with this secret token will be accepted
      https.addHeader("Authorization", secureToken);

      // Assemble the data to be sent
      String httpsRequestData = "uid=" + uid + "&RFID=" + actualRFID + "&candidateID=" + String(candidateID);

      // Get the response code after executing a POST
      int httpsResponseCode = https.POST(httpsRequestData);

      // If OK then read the payload
      if (httpsResponseCode == 200) {
        String payload = https.getString();
        Serial.println(payload);

        // Check if this account has already voted
        if(!payload.equals("Already Voted!")){
          digitalWrite(GREEN_LED, HIGH);
          delay(LED_TIMING);
          digitalWrite(GREEN_LED, LOW);
        } else {
          digitalWrite(RED_LED, HIGH);
          delay(LED_TIMING);
          digitalWrite(RED_LED, LOW);
        }

        // Updating the state so we switch to authentication
        state = true;
      }
      else {
        Serial.print("Error code: ");
        Serial.println(httpsResponseCode);
        digitalWrite(RED_LED, HIGH);
        delay(LED_TIMING);
        digitalWrite(RED_LED, LOW);
      }

      // Close the connection
      https.end();
    }
    else {
      Serial.println("WiFi Disconnected");
    }
}
```

Authentication function

```
// This function assigns a Blockchain account given a valid RFID ID
void authenticate(){
    if (rfid.PICC_IsNewCardPresent()) { // read a new RFID tag
     if (rfid.PICC_ReadCardSerial()) {
       char RFID[8] = "";

       // Transform the RFID tag to a Char array
       takeRFIDtoCharArray(rfid.uid.uidByte, rfid.uid.size, RFID);

       actualRFID = "";

       for(int i = 0; i < 8; i++){
           actualRFID += RFID[i];
       }

       if(WiFi.status()== WL_CONNECTED){

         // There we forge the HTTPS POST
         HTTPClient https;
         String serverPath = serverName + "/authenticate";
         https.begin(client, serverPath.c_str());
         https.addHeader("Content-Type", "application/x-www-form-urlencoded");

         // We add this token called "Authorization" because in the back-end of the webserver only
         // request with this secret token will be accepted
         https.addHeader("Authorization", secureToken);

         // Assemble the data to be sent
         String httpsRequestData = "RFID=" + actualRFID;

         // Get the response code after executing a POST
         int httpsResponseCode = https.POST(httpsRequestData);

         // If OK then read the payload
         if (httpsResponseCode == 200) {
           String payload = https.getString();

           // Assign the Blockchain ID given by the Flask Web Server
           actualBlockchainID = payload;
           digitalWrite(GREEN_LED, HIGH);
           delay(LED_TIMING);
           digitalWrite(GREEN_LED, LOW);

           // Updating the state so we switch to vote
           state = false;
         } else {
           Serial.print("Error code: ");
           Serial.println(httpsResponseCode);
           digitalWrite(RED_LED, HIGH);
           delay(LED_TIMING);
           digitalWrite(RED_LED, LOW);
           exit(-1);
         }

         // Close the connection
         https.end();
       } else {
         Serial.println("WiFi Disconnected");
       }
       rfid.PICC_HaltA();
       rfid.PCD_StopCrypto1();
     }
   }
}
```

RFID to Array

```
// This function transforms the RFID identifier to a Char Array
void takeRFIDtoCharArray(byte array[], unsigned int len, char buffer[])
{
   for (unsigned int i = 0; i < len; i++)
   {
      byte nib1 = (array[i] >> 4) & 0x0F;
      byte nib2 = (array[i] >> 0) & 0x0F;
      buffer[i*2+0] = nib1  < 0xA ? '0' + nib1  : 'A' + nib1  - 0xA;
      buffer[i*2+1] = nib2  < 0xA ? '0' + nib2  : 'A' + nib2  - 0xA;
   }
   buffer[len*2] = '\0';
}
```

Flask server

Flask server

```
from web3 import Web3
import json
from flask import Flask, request, jsonify
import mysql.connector as sqlconnect
from flask_sslify import SSLify

# Inizialize Database connection
DataBase = sqlconnect.connect(
    host="localhost",
    user="root",
    password="Votazioneseggio2022!",
    database="Seggio"
)
Cursor = DataBase.cursor()

# Shared secret between the Arduino and the server
SecureToken = "R3,u=t?_~LRrPycS"

# Force all the connection on HTTPS
app = Flask(__name__)
sslify = SSLify(app)

# Inizialize the connection with Ganache
provider = Web3.HTTPProvider('http://127.0.0.1:7545')
w3 = Web3(provider)

# Opens the JSON and takes the ABI of the Smart Contract SecureVoting.sol
with open("SecureVoting.json") as f:
    info_json = json.load(f)
abi = info_json["abi"]

# Inizialize the smart contract based on its address
secureVotingAddress = "0xd360F1924102fAac7EaB1EB83F7204Cab166055F"
contract = w3.eth.contract(address=secureVotingAddress, abi=abi)
```

Authenticate and checks

```python
# This function controls if the request has the secret shared token and it's performed before each function
@app.before_request
def verify_token():
    auth_header = request.headers.get("Authorization")
    if auth_header:
        if auth_header != SecureToken:
            return jsonify({"error": "You're not authorized"}), 401
    else:
        return jsonify({"error": "Missing Authorization header"}), 401

# This function checks if the server can comunicate with Ganache
@app.route('/isConnectedGanache', methods=['GET'])
def isConnected():
    try:
        return str(w3.isConnected())
    except:
        return jsonify({"error": "isConnect function does not work properly"}), 500

# This function returns a Blockchain ID given an RFID ID
@app.route('/authenticate', methods=['POST'])
def authenticate():
    try:
        payload = request.get_data().decode('utf-8')
        RFID = payload.split('=')[1]
        Cursor.execute("SELECT BlockchainID FROM Votante WHERE RFID = (%s)", (RFID,))
        for element in Cursor.fetchone():
            actualBlockchainID = element
        return str(actualBlockchainID)
    except:
        return jsonify({"error": "authenticate function does not work properly"}), 500
```

Vote and main

```python
# This function allows to vote given a Blockchain account ID.
# You can't vote more than one time.
@app.route('/vote', methods=['POST'])
def vote():
    try:
        payload = request.get_data().decode('utf-8').split('&')
        uid, RFID, candidateid = payload
        uid = uid.split('=')[1]
        candidateid = candidateid.split('=')[1]
        RFID = RFID.split('=')[1]
        for account in w3.eth._get_accounts():
            if account == uid:
                actualAccount = account
        Cursor.execute("SELECT alreadyVoted FROM Votante WHERE RFID = (%s)", (RFID,))
        for element in Cursor.fetchone():
            result = element
        if result == 1:
            return "Already Voted!"
        else:
            contract.functions.vote(str(uid), int(candidateid)).transact({"from": actualAccount})
            Cursor.execute("UPDATE Votante SET alreadyVoted = 1 WHERE RFID = (%s)", (RFID,))
            DataBase.commit()
            return str(contract.functions.totalVotes(int(candidateid)).call())
    except:
        return jsonify({"error": "vote function does not work properly"}), 500

# Starts the server and loads the certificates for HTTPS
if __name__ == '__main__':
    app.run(ssl_context=('cert/certificate.pem', 'cert/privatekey.pem'), host='0.0.0.0', port=443)
```

MySQL

Create DB

```
import mysql.connector as sqlconnect

DataBase = sqlconnect.connect(
    host="localhost",
    user="root",
    password="Votazioneseggio2022!"
)

Cursor = DataBase.cursor()

Cursor.execute("DROP DATABASE IF EXISTS Seggio")

Cursor.execute("CREATE DATABASE Seggio")
print("Database creato!")

Cursor.execute(("USE Seggio"))

TableName = "CREATE TABLE Votante ( RFID VARCHAR(8), BlockchainID VARCHAR(42), alreadyVoted Bit(1));"

Cursor.execute(TableName)
print("Tabella Votante creata!")
```

Populate DB

```
import mysql.connector as sqlconnect

DataBase = sqlconnect.connect(
    host="localhost",
    user="root",
    password="Votazioneseggio2022!",
    database="Seggio"
)

Cursor = DataBase.cursor()

sql = "INSERT INTO Votante (RFID, BlockchainID, alreadyVoted) VALUES (%s, %s, %s)"
val = [("9A4405B1", "0x09A7289877af2F7A69c6875D96d8F83aF6a5dCc3", 0),
       ("8C35D337", "0xbe109a0F6765bB5C30f02CE33291C3f491B9993A", 0),
       ("6A750BB1", "0xC42a285a7C4D9630320a3A2Dd7642219BA31bAd3", 0),
       ("FA4FF6B0", "0xfa4D4da0351950d82c65FFda67b3CbB7d43Ed578", 0),
       ("83741AAA", "0x4a4418eeb4a05214E2A8BD298B176288983f3227", 0),
       ("233E92A9", "0xcbD2e874a33B31454f1f697677B65c682620E5DE", 0),
       ("436ECDA9", "0x8C07Beb67A8FD8FDD29953EFe145C0F88Ff0FF07", 0),
       ("13687EA9", "0xaCEFC97EE71E14dDC00e7c296Ae9aeD8969c0c1C", 0)]

Cursor.executemany(sql, val)
DataBase.commit()

print(Cursor.rowcount, "righe inserite!")

# disconnessione
DataBase.close()
```

Init Ganache

```python
from web3 import Web3
import json

# Inizialize the connection with Ganache
provider = Web3.HTTPProvider('http://127.0.0.1:7545')
w3 = Web3(provider)

# Opens the JSON and takes the ABI of the Smart Contract SecureVoting.sol
with open("SecureVoting.json") as f:
    info_json = json.load(f)
abi = info_json["abi"]

# Inizialize the smart contract based on its address
secureVotingAddress = "0xd360F1924102fAac7EaB1EB83F7204Cab166055F"
contract = w3.eth.contract(address=secureVotingAddress, abi=abi)

# Takes the first account, the one who deployed the Smart Contract and the only one
# who can add candidates
ownerAccount = w3.eth._get_accounts()[0]

# Addes 4 candidates to permit the voting
contract.functions.addCandidate("Gerardo", "Lega Nord").transact({"from": ownerAccount})
contract.functions.addCandidate("Giovanni", "Partito Democratico").transact({"from": ownerAccount})
contract.functions.addCandidate("Mario", "Europa+").transact({"from": ownerAccount})
contract.functions.addCandidate("Antonio", "Movimento 5 Stelle").transact({"from": ownerAccount})

# Prints the number of candidates
numberOfCandidates = contract.functions.getNumOfCandidates().call()
print("Number of Candidates: " + str(numberOfCandidates))

# Print the candidates informations
i = 0
while i < numberOfCandidates:
    print(contract.functions.getCandidate(i).call())
    i = i + 1
```

Test

Main test

```
import requests

url = 'https://192.168.1.153:443/vote'
data = {
    'uid': '0x4c72fD9A313E8C50bcE0E513b55D35f1568481da',
    'RFID': '8C35D337',
    'candidateID': '0'
}

response = requests.post(url, data=data, verify=False)

print(response.text)
print("Status Code", response.status_code)
```

04

Security and vulnerability

Security, privacy and vulnerability overview

Security

Security

1 - **System security**: data is retrieved and sent **if and only if** the RFID reader reads the correct card.
2 - **Communication security**: data is transferred between the client and server through a TLS 1.3 connection.
3 - The usage of TLS on top of TCP provides **Confidentiality** and **Integrity.** The usage of certificates ensures **Authentication.**

Vulnerabilities

Main vulnerabilities

1 - **Credential storage**: The database password and the shared secret token are stored in the code, which is a security risk if the code is leaked or the server is compromised.
2 - **Use of HTTP**: The code uses HTTP instead of HTTPS to communicate with Ganache, which makes it vulnerable to man-in-the-middle attacks and information theft.
3 - **Hardcoded addresses**: The smart contract address is hardcoded in the code, which means that if the address changes, the code will stop working.
4 - **Lack of encryption**: The code does not use encryption for storing data, making it vulnerable to data theft and tampering.

Thank you!

Giovanni Rapa