Zugangsbeschränkung zu Geräten per Contactless Card mit der NodeMCU und dem RC522 Modul vierter Teil – Down the Rabbit Hole

Im heutigen Blog geht es, wie versprochen, "down the Rabbit Hole" und wir beschäftigen uns mit den Interna der MiFare Proximity Integrated Contactless Cards (PICC) Reihe. Wir gehen auf die Besonderheiten ein, und schauen und die Funktionen genauer an.

Viele Card Reader- Projekte, mit dem MF522 Chipsatz als Basis uns dem Arduino die ich persönlich schon gesehen habe, nutzen leider weder in Sicherheitstechnisch noch in Möglichkeiten das aus, was uns die Mifare- Classic Karte an Funktionen bietet.

Diese Projekte beschränken sich darauf, die frei für jeden Kartenleser und Handys lesbare Unique ID (UUID) der Karte zu lesen und diese gegen eine Liste von erlaubten UUID's gegen zu prüfen. Ist diese in der Liste enthalten, wird die Karte als gültig angesehen.

Auch in unseren vorherigen Teilen machen wir von diesem Prinzip gebrauch. Dies wollen wir mit unserem heutigen Blogeintrag ändern und Daten abgesichert auf unserer MiFare Classic Karte ablegen.

Durch den in diesem Projekt genutzten Kartentyp "MiFare Classic" (bitte auch diesen Hinweis beachten) können wir eigene, von uns vorgegebene Informationen, aus der Karte speichern und diese gegen unautorisiertes auslesen oder verändern schützen.

Dazu sind vom Hersteller bereits auf der Karte 16 Sektoren (0-15) mit je 4 * 16 Bytes vorhanden bei denen, mit Ausnahme des Sektors 0 , 3 * 16 Byte frei beschrieben werden können. 16 Bytes eines jeden Sektors werden Sektor Trailer genannt und zur Ablage der 2 Sektorenschlüssel und zur Zugriffsmatrixdefinition genutzt.

Die 16 Bytes eines Sektor Trailer sind wie folgt aufgeteilt:

- 6 Bytes - erster Sektorenschlüssel
- 6 Bytes - zweiter Sektorenschlüssel
- 4 Bytes - Zugriffsberechtigungsdefinition

Wir wollen auf unserer Karte nun den Vornamen und den Nachnamen von uns verschlüsselt speichern. Dazu definieren wir einen Sektor (1-15) über die Konstante ... den wir für diesen Zweck verwenden wollen. Ich habe im Code den Sektorenblock 1 gewählt.

Der Vorname soll im zweiten 16 Bytes Feld, und der Nachname im drittem 16 Bytes Feld stehen. Das erste 16 Bytes Feld ist für zukünftige Erweiterungen reserviert.

Desweitern brauchen wir eine Schlüsseldefinition, die wir in der Struktur MiFareClassicKeyTable hinterlegen.

Bitte ändert diesen Block (2x 6 Bytes) beim Nachbau unbedingt mit eigenem Schlüsselmaterial ab und speichert euch dieses an einem sicheren Ort, da sonst jeder, der dieses Projekt kennt mit dem im Code im Klartext genannten Schlüsselmaterial eure Karte lesen und eigene gültige! Karten für eure Gerätesteuerung anlegen kann.

Spoiler: Außerdem werden wir dieses Schlüsselmaterial bei einer späteren Fortsetzung der Reihe wieder benötigen. Bitte beachtet weiterhin, das ihr die Karte beschreibt und Sicherheitsoptionen ändert. Dadurch ist die Karte evtl. nicht mehr für andere Einsatzzwecke nutzbar!

Wir laden nach Änderung des Schlüsselmaterials folgenden Code auf unseren ESP hoch:

```
#include <SPI.h>
#include <MFRC522.h>
#include <ESP8266WiFi.h>
//#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>
#include <EEPROM.h>
#include <FS.h>          // Include the SPIFFS library

#define RST_PIN     5     // SPI Reset Pin (D1  Ausgang)
#define RELAIS_PIN  16    // Relais (D0  Ausgang) [LOW Aktiv] - Auch interne LED nahe USB Port
#define SS_PIN      15    // SPI Slave Select Pin

#define RGBLED_R    2     // Rot (D4  Ausgang)
#define RGBLED_G    0     // Grün (D3  Ausgang) - Auch interne LED auf dem ESP Modul
#define RGBLED_B    4     // Blau (D2  Ausgang)

#define WiFiPwdLen   25   // Maximale WiFi Passwortlänge
#define STANameLen 20     // Maximale WiFi SSIDlänge
#define ESPHostNameLen 20 // Maximale Anzahl Zeichen ESPHostName

#define KEYA true         //PICC Flag Definition
#define KEYB false        //PICC Flag Definition

#define LED_BUILTIN 16
#define PIN_WIRE_SDA 4
#define PIN_WIRE_SCL 5

#define USED_Sector 1  // Kartensektor, der für die Authentifizierungs - und Konfigurationsdaten evrwendet wird ( gültig: 1-15)

ADC_MODE(ADC_TOUT);   // Analogeingang A0 auf extern konfigurieren. ADC_TOUT (for external voltage), ADC_VCC (for system voltage).
MFRC522 mfrc522(SS_PIN, RST_PIN);   // Instanz des MFRC522 erzeugen
MFRC522::MIFARE_Key key;
ESP8266WebServer server(80);        // Web Server Instanz erzeugen

struct WiFiEEPromData
 {
   char ESPHostName[ESPHostNameLen];
   char APSTAName[STANameLen]; // STATION /AP Point Name TO Connect, if definded
   char WiFiPwd[WiFiPwdLen]; // WiFiPAssword, if definded
   char ConfigValid[3]; //If Config is Vaild, Tag "TK" is required"
 };

struct MiFareClassicKeyTable
 {
  byte Key_A[6] = {0x22,0x44,0xFA,0xAB,0x90,0x11};   // Schlüssel für PICC Karte bitte ändern.
  byte Key_B[6] = {0xFE,0xE1,0xAA,0x3D,0xDF,0x37};   // Schlüssel für PICC Karte bitte ändern.
  char ConfigValid[3]; //If Config is Vaild, Tag "TK" is required"
 };
```

```
MiFareClassicKeyTable MiFareClassicKey;
WiFiEEPromData MyWiFiConfig;

// Global genutzte Variablen
bool Result  = false;
bool LearnNewCard = false;
bool EraseCard = false;
bool ExpirationDateActive = false;
String Surname;
String Givenname;
String ExpirationDate;
String temp;
unsigned long SessionID;
unsigned long PCD_ServiceCall_Handler = 0;
unsigned long PCD_WatchDog_Handler = 0;
uint8_t DataBuffer[18]  = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 };


void setup()
  {
  pinMode(RST_PIN,OUTPUT);
  digitalWrite(RST_PIN,HIGH);
  pinMode(RELAIS_PIN,OUTPUT);
  pinMode(RGBLED_R,OUTPUT);
  pinMode(RGBLED_G,OUTPUT);
  pinMode(RGBLED_B,OUTPUT);
  digitalWrite(RELAIS_PIN,HIGH);    // Relais inaktiv
  SetRGBLed(255,0,255,false);       //Led Farbe Lila Initalisierung beginnen
  Serial.begin(115200);             // Serielle Kommunikation mit dem PC mit 115200 Baud initialisieren
  Serial.println("");
  temp = "ATSN:"+ String(ESP.getChipId());
  Serial.println(temp);
  // Serial.print("ADC Value:");Serial.println(analogRead(A0));
  SPI.begin();                      // Initialisiere SPI Kommunikation
  PCDHardReset();
  SessionID = millis();
  Result  = startWiFiClient();
  InitalizeHTTPServer();
  SetRGBLed(0,0,255,false);       //Led Farbe Blau Initalisierung abgeschlossen
  //ESP.wdtEnable(WDTO_4S);             // Starte Watchdog
  }


// ****************** Start Helper/ Optimization Functions ********************************

void SetRGBLed(byte RedValue,byte GreenValue,byte BlueValue,boolean SlowFade)  //Funkion zur Steuerung der RGB Led
{
  digitalWrite(RGBLED_R,LOW);
  digitalWrite(RGBLED_G,LOW);
  digitalWrite(RGBLED_B,LOW);
  if (RedValue == 255)   { digitalWrite(RGBLED_R,HIGH); }
  if (GreenValue == 255) { digitalWrite(RGBLED_G,HIGH); }
  if (BlueValue == 255)  { digitalWrite(RGBLED_B,HIGH); }
}

// ****************** Stop Helper/ Optimization Functions *********************************

// ****************** Start Functions Webserver ***********************************************

//Cookie Basisroutinen basieren auf GIT Auszug:
//https://github.com/esp8266/ESPWebServer/blob/master/examples/SimpleAuthentification/SimpleAuthentification.ino
bool is_authentified()
{
   if (server.hasHeader("Cookie")){
     // Cookie gefunden
     temp = server.header("Cookie");
     //Serial.println(temp);
     String SessionStr = String(ESP.getChipId()) + "=" + String(SessionID);
     yield();
     if (temp.indexOf(SessionStr) != -1) {
       // Web Authentification erfolgreich
       temp = "";
       return true;
     }
   }
   // Web Authentification fehlgeschlagen
```

```
  temp = "";
  SessionID = millis();
  return false;
}

void handleLogin(){
 String msg;
 //String cookie = server.header("Cookie");
 //Serial.println(cookie);
 if (server.hasArg("DISCONNECT")){
  //Disconnection Benutzers;
  server.sendHeader("Location","/login");
  server.sendHeader("Cache-Control","no-cache");
  SessionID = millis();
  temp = String(ESP.getChipId()) + "= NA ; HttpOnly ; SameSite=Strict";
  server.sendHeader("Set-Cookie",temp);
  temp = "";
  server.send(301);
  yield();
  return;
 }
 if (server.hasArg("USERNAME") && server.hasArg("PASSWORD")){
  temp = String(ESP.getChipId());
  if (server.arg("USERNAME") == "admin" &&  server.arg("PASSWORD") == temp ){
   server.sendHeader("Location","/");
   server.sendHeader("Cache-Control","no-cache");
   SessionID = millis();
   temp = String(ESP.getChipId()) + "=" + String(SessionID) + "; HttpOnly ; SameSite=Strict";
   server.sendHeader("Set-Cookie",temp);
   temp = "";
   server.send(301);
   yield();
   return;
  }
 msg = "<script>alert('Falscher Benutzername oder falsches Passwort !');</script>";
 }
  CSS_Header_Template();
  yield();
  temp = "<head><title>Login</title></head><body><DIV ALIGN=CENTER>";
  server.sendContent(temp);
  temp = "<h2>Anmeldung an Kartenleser RC522</h2><body><br><br><table border=0 bgcolor=black><tr><th><DIV
ALIGN=RIGHT>";
  server.sendContent(temp);
  temp = "<form action='/login' method='post'>Benutzername: <input type=text Name='USERNAME' Size=17 required><br>";
  server.sendContent(temp);
  temp = "Passwort: <input type=password Name='PASSWORD' Size=17 required><br><br><br><button type='submit' ";
  server.sendContent(temp);
  temp = "name='Login_Button' value='1' style='height: 30px; width: 100px'
>Login</button><br></th></tr></form></DIV></table>";
  server.sendContent(temp);
  temp = "<br><SMALL>Damit die Anmeldung funktioniert, sind Cookies für diese Webseite zu erlauben.</SMALL>";
  server.sendContent(temp);
  temp = msg + "</DIV></body></HTML>";
  server.sendContent(temp);
  temp = "";
}

void handleNotFound()
 {
 SessionID = millis();
 temp = "Seite nicht gefunden.\n\n";
 temp += "URI: ";
 temp += server.uri();
 temp += "\nMethod: ";
 temp += (server.method() == HTTP_GET)?"GET":"POST";
 temp += "\nArguments: ";
 temp += server.args();
 temp += "\n";
 for (uint8_t i=0; i<server.args(); i++){
  temp += " " + server.argName(i) + ": " + server.arg(i) + "\n";
 }
 yield();
 server.send(404, "text/plain", temp);
 temp = "";
 }
```

```
void handleErasePICC()
{
  if (!is_authentified())
    {
    server.sendHeader("Location","/login");
    server.sendHeader("Cache-Control","no-cache");
    server.send(301);
    yield();
    return;
    }
  CSS_Header_Template();
  yield();
  temp = "<head><title>Kartenleser RC522</title></head><body>";
  server.sendContent(temp);
  HtmlNavStructure();
  temp = "<script>alert('Bitte JETZT die zu löschende Karte vor den Leser halten!');</script>";
  server.sendContent(temp);
  yield();
  SetRGBLed(0,255,255,false);       //Led Farbe cyan Programmierungsmodus
  EraseCard = true;
  temp = "</body></html>";
  server.sendContent(temp);
  server.client().stop();
  temp = "";
}


void handleNewPICC()
{
  if (!is_authentified())
    {
    server.sendHeader("Location","/login");
    server.sendHeader("Cache-Control","no-cache");
    server.send(301);
    return;
    }
  if (server.hasArg("Surname") && server.hasArg("Givenname"))
  {
    Surname = server.arg("Surname");
    Givenname = server.arg("Givenname");
    ExpirationDate = server.arg("ExpDate");
    if (server.hasArg("ExpDateOption")) { ExpirationDateActive = true; } else { ExpirationDateActive = false; }
    temp = "<script>alert('Bitte JETZT die neue Karte vor den Leser halten!');</script>";
    server.sendContent(temp);
    SetRGBLed(255,255,0,false);       //Led Farbe Gelb Programmierungsmodus
    LearnNewCard = true;
    yield();
    return;
  }
  CSS_Header_Template();
  yield();
  temp = "<head><title>Kartenleser RC522</title></head><body>";
  server.sendContent(temp);
  HtmlNavStructure();
  temp = "";
  temp = "<br><br><br><br><table border=0 ALIGN=CENTER><th>";
  server.sendContent(temp);
  temp = "<table border=1 bgcolor = black><form action='/newPICC' method='post'>";
  server.sendContent(temp);
  temp = "<tr><th>Karteninhaber:<br><div ALIGN=RIGHT>";
  server.sendContent(temp);
  temp = "Vorname: <input type=text Name='Surname' Size=17 maxlenght=16 placeholder='Max' required><br>";
  server.sendContent(temp);
  temp = "Nachname: <input type=text Name='Givenname' Size=17 maxlenght=16 placeholder='Mustermann' required><br>";
  server.sendContent(temp);
  temp = "</div></th><th>Kartenmetadaten:<br><DIV ALIGN=RIGHT>";
  server.sendContent(temp);
  temp = "<input Name='ExpDateOption' TYPE=checkbox VALUE=1 >Ablaufdatum:<input type=date Name='ExpDate' Size =
17 >";
  server.sendContent(temp);
  temp = "<br><th><tr><th></table><br>";
  server.sendContent(temp);
  temp = "<button type='submit' name='NewCard' value='1' style='height: 30px; width: 200px' >Neue Smartcard
erstellen</button>";
  server.sendContent(temp);
  temp = "<br></form></tr></th></table>";
```

```
  server.sendContent(temp);
  temp = "</body></html>";
  server.sendContent(temp);
  server.client().stop();
  yield();
  temp = "";
}

void handleRoot()
{
  if (!is_authentified()){
    server.sendHeader("Location","/login");
    server.sendHeader("Cache-Control","no-cache");
    server.send(301);
    return;
    }
  // HTML Content
  CSS_Header_Template();
  yield();
  temp = "<head><title>Kartenleser RC522</title></head><body>";
  server.sendContent(temp);
  HtmlNavStructure();
  temp = "<div ALIGN=CENTER><br><br><br><br><BIG>Willkommen auf der Smartkartenleser RC522
Webseite.</BIG><br>";
  server.sendContent(temp);
  temp = "Resetgrund: " + String(ESP.getResetReason()) + "<br>";
  server.sendContent(temp);
  temp = "Freier Heapspeicher: " + String(ESP.getFreeHeap()) + " Bytes<br>";
  server.sendContent(temp);
  temp = "Int. Flash: " + String(ESP.getFlashChipRealSize()) + " Bytes<br>";
  server.sendContent(temp);
  Result = mfrc522.PCD_PerformSelfTest();
  mfrc522.PCD_Init();                    // Initialisiere MFRC522 Lesemodul
  mfrc522.PCD_SetAntennaGain(mfrc522.RxGain_max); // Setzt Antenne auf max. Empfang
  mfrc522.PCD_AntennaOn();
  yield();
  if (Result) {temp = "RC522 PCD-Status: OK<br>"; } else {temp = "RC522 PCD-Status: Fehler!<br>"; }
  server.sendContent(temp);
  temp = "CPU ID: " + String(ESP.getChipId()) + " @ " + String(ESP.getCpuFreqMHz()) + " MHz<br>";
  server.sendContent(temp);
  temp = "<br>Sie sind erfolgreich angemeldet !<br><br><form action='/login' method='get'>";
  server.sendContent(temp);
  temp = "<button type='submit' name='DISCONNECT' value='YES' style='height: 30px; width: 200px' >Logout</button>";
  server.sendContent(temp);
  temp = "</form></div></body></html>";
  server.sendContent(temp);
  if (server.hasArg("Reboot") )  // Reboot System
    {
    //ESP.wdtFeed();
    ESP.wdtDisable();
    temp = "<script>alert('Das System startet JETZT neu.');</script>";
    server.sendContent(temp);
    server.client().stop();
    yield();
    temp = "";
    ESP.reset();
    delay(4000);
    }
  server.client().stop();
  temp = "";
}

void CSS_Header_Template() // Formatvorlage für alle internen ESP Webseiten. https://wiki.selfhtml.org/wiki/CSS
  {
   server.setContentLength(CONTENT_LENGTH_UNKNOWN);
   temp = "";
   server.send (200, "text/html", temp);
   temp = "<!DOCTYPE HTML PUBLIC '-//W3C//DTD HTML 4.01 Transitional//EN'><html lang='de'><meta charset='UTF-
8'>";
   server.sendContent(temp);
   temp = "<style type='text/css'>*{margin: 0;padding: 0;}body{background:black;color:darkorchid;font-size: 16px;";
   server.sendContent(temp);
   temp = "font-family: sans-serif,arial;}.nav{width: 1300px;height: 30px;margin: 0 auto;border-radius: 5px;}";
   server.sendContent(temp);
   temp = "ul li{list-style: none;width: 200px;line-height: 60px;position: relative;background: darkorchid;";
   server.sendContent(temp);
   temp = "box-shadow: 0px 2px 5px 0px grey;text-align: center;float: left;background-color: #010000;}ul li ul{";
```

```
  server.sendContent(temp);
  temp = "position: absolute;}.nav > ul > li:nth-of-type(1){border-radius: 5px 0px 0px 5px;}.nav > ul > li:nth-of-type(5)";
  server.sendContent(temp);
  temp = "{border-radius: 0px 5px 5px 0px;}ul li a{color: rgb(182, 18, 18);width: 200px;height: 58px;display: inline-block;";
  server.sendContent(temp);
  temp = "text-decoration: none;}ul li a:hover{font-weight: bold;border-bottom: 2px solid #fff;}ul li ul{display: none;}";
  server.sendContent(temp);
  temp = ".nav ul li:hover ul{display: block;}.fa{margin-right: 5px;}.container{width: 1000px;height: 200px;";
  server.sendContent(temp);
  temp = "margin: 0 auto;padding:20px 20px;}@media screen and (max-width: 480px){header{width: 100%;}";
  server.sendContent(temp);
  temp = ".nav{display: none;width: 100%;height: auto;}ul li{width: 100%;float: none;}ul li a{width: 100%;";
  server.sendContent(temp);
  temp = "display: block;}ul li ul{position: static;}ul li ul li a{background: #222;}.fa-list.modify{display: block;}";
  server.sendContent(temp);
  temp = ".container{width: 100%;height: auto;}body{overflow-x:hidden;}}</style>";
  server.sendContent(temp);
  temp = "";
 }


void HtmlNavStructure()
 {
 temp = "<div class='menu'><nav class='nav'><ul>";
 server.sendContent(temp);
 temp = "<li><a href='#'>System</a>";
 server.sendContent(temp);
 temp = "<ul><li><a href='/'>Information</a></li>";
 server.sendContent(temp);
 temp = "<li><a href='/?Reboot=YES'>Neustart</a></li>";
 server.sendContent(temp);
 temp = "</ul>";
 server.sendContent(temp);
 temp = "</li><li><a href='#'>PICC</a>";
 server.sendContent(temp);
 temp = "<ul><li><a href='/newPICC'>Neue Karte erstellen</a></li>";
 server.sendContent(temp);
 temp = "<li><a href='/erasePICC'>Karte löschen</a></li></ul>";
 server.sendContent(temp);
 temp = "</li>";
 //temp = "</li><li><a href='#'>Ereignisprotokoll</a></li>";
 server.sendContent(temp);
 temp = "</ul></nav></div>";
 server.sendContent(temp);
 temp = "";
 }


void InitalizeHTTPServer()
 {
 bool initok = false;
 const char * headerkeys[] = {"User-Agent","Cookie"} ; //Header zum Tracken
 size_t headerkeyssize = sizeof(headerkeys)/sizeof(char*); //Header zum Tracken
 server.on("/", handleRoot);
 server.on("/login", handleLogin);
 server.on("/newPICC", handleNewPICC);
 server.on("/erasePICC", handleErasePICC);
 server.onNotFound ( handleNotFound );
 server.collectHeaders(headerkeys, headerkeyssize );// Server anweisen, diese zu Tracken
 server.begin(); // Web server start
 }



// ****************** End Functions Webserver *********************************************


// ****************** Start Functions WiFi Management ************************************
// Funktion von https://www.az-delivery.de/blogs/azdelivery-blog-fur-arduino-und-raspberry-pi/wps-mit-dem-esp8266?ls=de
bool startWPS()
{
 bool wpsSuccess = WiFi.beginWPSConfig();
 if(wpsSuccess) {
    // Muss nicht immer erfolgreich heißen! Nach einem Timeout ist die SSID leer
    String newSSID = WiFi.SSID();
     if(newSSID.length() > 0) {
      // Nur wenn eine SSID gefunden wurde waren wir erfolgreich
```

```
        yield();
        Serial.println("ATWPS:OK");
        saveCredentials(); // Save Credentials to EEPROM
      } else {
        Serial.println("ATWPS:NOK");
      }
  }
  return wpsSuccess;
}

bool startWiFiClient()
{
  bool WiFiClientStarted = false;
  size_t A0_ADCValue = 0;
  byte i = 0;
  byte connRes = 0;
  Serial.setDebugOutput(false);  // Zu Debugzwecken aktivieren.
  WiFi.hostname("CrdRdr41667");
  WiFi.softAPdisconnect(true);
  WiFi.disconnect();
  WiFi.mode(WIFI_STA);
  if(loadCredentials())
    {
     WiFi.begin(MyWiFiConfig.APSTAName, MyWiFiConfig.WiFiPwd);
     while (( connRes != 3 ) and( connRes != 4 ) and (i != 30))  //if connRes == 0  "IDLE_STATUS - change Statius"
      {
      i++;
      // Serial.print("."); // Connect vorgang auf der seriellen Schnittstelle beobachten
      //ESP.wdtFeed();
      delay(500);
      yield();
      connRes  = WiFi.waitForConnectResult();
      }
     if (connRes == 4 ) { // if password is incorrect
      Serial.println("ATWIFI:PWDERR");
      WiFi.disconnect();
      }
     if (connRes == 6 ) { //  module is not configured in station mode
      Serial.println("ATWIFI:STAERR");
      WiFi.disconnect();
      }
    }
  if(WiFi.status() == WL_CONNECTED)
    {
    //ESP.wdtFeed();
    Serial.print("ATIP:");
    Serial.println(WiFi.localIP());
    WiFi.setAutoReconnect(true); // Set whether module will attempt to reconnect to an access point in case it is disconnected.
    // Setup MDNS responder
    if (!MDNS.begin("CrdRdr41667"))
      {
      Serial.println("ATMDNS:NOK");
      } else { MDNS.addService("http", "tcp", 80); }
    WiFiClientStarted = true;
    } else
    {
    A0_ADCValue = analogRead(A0);
    //Wir waren nicht erfolgreich, daher starten wir WPS, wenn WPS Taster an A0 während des Resets gedrückt ist
    if (A0_ADCValue > 499)
      {
        if(startWPS())
          {
            //ESP.wdtFeed();
            delay(500);
            WiFi.disconnect();
            WiFi.mode(WIFI_STA);
            WiFi.begin(WiFi.SSID().c_str(), WiFi.psk().c_str());
            //ESP.wdtFeed();
            WiFiClientStarted = true;
          } else
          {
            WiFiClientStarted = false;
            WiFi.disconnect();
          }
      } else
      {
        WiFi.disconnect();
```

```
    }
  }
  //WiFi.printDiag(Serial);       // Zu Debugzwecken aktivieren.
  return WiFiClientStarted;
}
// ******************* END Functions WiFi Management ************************************

// ******************* Start Functions Store WiFi Credentials to EEPROM ******************
bool loadCredentials()
{
 bool RetValue;
 EEPROM.begin(512);
 EEPROM.get(0,MyWiFiConfig);
 EEPROM.end();
 if (String(MyWiFiConfig.ConfigValid) == "TK")
  {
    RetValue = true;
  } else
  {
    RetValue = false; // WLAN Settings nicht gefunden.
  }
  //ESP.wdtFeed();
  return RetValue;
}

void saveCredentials() // Speichere WLAN credentials auf EEPROM
{
  size_t i;
  for (i = 0 ; i < sizeof(MyWiFiConfig) ; i++) // Loeschen der alten Konfiguration
     {
      EEPROM.write(i, 0);
     }
  for (i = 0 ; i < STANameLen  ; i++) // Loeschen der alten Konfiguration
     {
      MyWiFiConfig.WiFiPwd[i] = 0;
     }
  for (i = 0 ; i < WiFiPwdLen ; i++) // Loeschen der alten Konfiguration
     {
      MyWiFiConfig.APSTAName[i] = 0;
     }
  temp = WiFi.SSID().c_str();
  i = temp.length();
  temp.toCharArray(MyWiFiConfig.APSTAName,i+1);
  temp = WiFi.psk().c_str();
  i = temp.length();
  temp.toCharArray(MyWiFiConfig.WiFiPwd,i+1);
  temp = "";
  strncpy(MyWiFiConfig.ConfigValid , "TK", sizeof(MyWiFiConfig.ConfigValid) );
  EEPROM.begin(512);
  EEPROM.put(0, MyWiFiConfig);
  EEPROM.commit();
  EEPROM.end();
  //ESP.wdtFeed();
}
// ******************* END Functions StoreCredentialsto EEPROM ***************************

// ******************* Start Functions CardServices  ************************************


void PCDHardReset()
{
  digitalWrite(RST_PIN,LOW);
  delay(200);
  digitalWrite(RST_PIN,HIGH);
  mfrc522.PCD_Reset();
  mfrc522.PCD_Init();                          // Initialisiere MFRC522 Lesemodul
  mfrc522.PCD_SetAntennaGain(mfrc522.RxGain_max); // Setzt Antenne auf max. Empfang
  mfrc522.PCD_AntennaOn();
}

boolean CardAuthenticate(boolean ABKey, byte Sector,byte ikey[6])
{
const byte sectorkeytable [16] = {3,7,11,15,19,23,27,31,35,39,43,47,51,55,59,63};
byte statusA;
statusA = 0;
for (int a = 0; a < 6;a++)
  {
```

```
  key.keyByte[a] = ikey[a];
  }
  // Key A
if (ABKey)
  {
  statusA = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, sectorkeytable[Sector], &key,
&(mfrc522.uid));
  if (statusA != MFRC522::STATUS_OK)
    {
    Serial.println("ATAUTH:ERR_A");
    return false;
    }
  }
  // Key B
  else if (not ABKey)
  {
  statusA = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_B,sectorkeytable[Sector], &key,
&(mfrc522.uid));
  if (statusA != MFRC522::STATUS_OK)
    {
    Serial.println("ATAUTH:ERR_B");
    return false;
    }
  }
return true;
}

// WriteData . uses Global Variable DataBuffer for Data Return
boolean CardDataWrite(byte Sector,byte block,byte value[16])
{
byte status;
byte writevector;
byte sectorkeytable [16] = {3,7,11,15,19,23,27,31,35,39,43,47,51,55,59,63};
writevector = Sector * 4 + block -1 ;
for (byte a = 0; a < 16; a++)
  {
  if (writevector ==  sectorkeytable[a])
    {
    // Serial.println("NAK");
    return false;
    }
  }
status = mfrc522.MIFARE_Write(writevector, value, 16);
if (status != MFRC522::STATUS_OK)
  {
  Serial.println("ATPCD:W_ERR");
  //Serial.println(mfrc522.GetStatusCodeName(status));
  return false;
  } else
  {
  // Serial.print("OK");
  return true;
  }
}

// Read Data - uses Global Variable DataBuffer for Data Return
boolean CardDataRead(byte Sector,byte block)
{
byte statusi;
byte readvector;
const byte sectorkeytable [16] = {3,7,11,15,19,23,27,31,35,39,43,47,51,55,59,63};
byte sized = 18;
readvector = Sector * 4 + block -1 ;
for (byte a = 0; a < 16; a++)
  {
  if (readvector ==  sectorkeytable[a])
    {
    Serial.println("ATPCD:R_ERR");
    return false;
    }
  }
statusi = mfrc522.MIFARE_Read(readvector, DataBuffer, &sized);
if (statusi != MFRC522::STATUS_OK)
  {
  Serial.println("ATPCD:R_ERR");
  return false;
  } else
```

```
  {
  return true;
  }
}

boolean ResetCardToDefault()
{
byte ikey[16];
byte status,i;
byte writevector;
const byte sectorkeytable [16] = {3,7,11,15,19,23,27,31,35,39,43,47,51,55,59,63};
writevector = sectorkeytable[USED_Sector];
if (CardAuthenticate(KEYB,USED_Sector,MiFareClassicKey.Key_B)) // Sector Autenticate for WRITE Access
    {
     for (i = 0; i <= 16; i++) { DataBuffer[i] = 0; }
     if (!(CardDataWrite(USED_Sector,1,DataBuffer))) { return false; }
     for (i = 0; i <= 16; i++) { DataBuffer[i] = 0; }
     if (!(CardDataWrite(USED_Sector,2,DataBuffer))) { return false; }
     for (i = 0; i <= 16; i++) { DataBuffer[i] = 0; }
     if (!(CardDataWrite(USED_Sector,3,DataBuffer))) { return false;}
    }
for (byte i = 0; i <= 16; i++) { ikey[i] = 255; }  //Load Default Key for all Sectors
ikey[6] = 0xFF; // Default Setting for Access Bits
ikey[7] = 0x07; //
ikey[8] = 0x80; //
ikey[9] = 0x69;
status = mfrc522.MIFARE_Write(writevector, ikey, 16);
if (status != MFRC522::STATUS_OK)
  {
  return false;
  }
return true;
}

boolean SetSectorAccessControl (byte Sector,byte Akey[6],byte Bkey[6])
{
byte ikey[16];
byte status;
byte writevector;
const byte sectorkeytable [16] = {3,7,11,15,19,23,27,31,35,39,43,47,51,55,59,63};
writevector = sectorkeytable[Sector];
ikey[0] = Akey[0];
ikey[1] = Akey[1];
ikey[2] = Akey[2];
ikey[3] = Akey[3];
ikey[4] = Akey[4];
ikey[5] = Akey[5];
ikey[6] = 0x78; // Data Block 0-3 Access Conditions: Key B write / Key A Read
ikey[7] = 0x77; // KEY A & KEY B & Acces Bits Write:Key B  / Key A Read Access Bits
ikey[8] = 0x88; // Calculator: http://calc.gmss.ru/Mifare1k/
ikey[9] = 0x69; // Fixer Wert - > default hex 69
ikey[10] = Bkey[0];
ikey[11] = Bkey[1];
ikey[12] = Bkey[2];
ikey[13] = Bkey[3];
ikey[14] = Bkey[4];
ikey[15] = Bkey[5];
status = mfrc522.MIFARE_Write(writevector, ikey, 16);
if (status != MFRC522::STATUS_OK)
  {
  Serial.println("ATPCD:W_KEY_ERR");
  return false;
  }else
  {
  return true;
  }
}

boolean CheckforDefaultCardKey ()
{
byte tkey[6];
boolean CardResult;
byte readvector;
byte statusi;
const byte sectorkeytable [16] = {3,7,11,15,19,23,27,31,35,39,43,47,51,55,59,63};
byte sized = 18;
for (byte i = 0; i <= 6; i++) { tkey[i] = 255; }  //Load Default Key for all Sectors
```

```
 CardResult = true;
 if (!CardAuthenticate(KEYA,USED_Sector,tkey)) { CardResult = false; };
 readvector = sectorkeytable[USED_Sector];
 statusi = mfrc522.MIFARE_Read(readvector, DataBuffer, &sized);
 if (statusi != MFRC522::STATUS_OK) { CardResult = false; }
 //if (!((DataBuffer[7] = 0x07) & (DataBuffer[7] = 0x80)))  { CardResult = false; };
 return CardResult;
 }

 boolean WriteNewMiFareClassicPICC ()
  {
  byte tkey[6];
  byte i,a;
  boolean CardResult;
  if (CheckforDefaultCardKey())
   {
   for (i = 0; i <= 6; i++) { tkey[i] = 255; }  //Load Default Key for all Sectors
   for (i = 0; i <= 16; i++) { DataBuffer[i] = 0; } // Clear Variable Buffer
   CardResult = true;
   if (CardAuthenticate(KEYA,USED_Sector,tkey)) // Sector  Autenticate
    {
    // Serial.println("Auth Sec 0 OK");
    if (Surname.length() > 15) { a = 15; } else { a = Surname.length();}
    if (Surname.length() > 0)
     {
     for (i = 0; i <= 16; i++) { DataBuffer[i] = 0; }
     for (i = 0; i <= a; i++) { DataBuffer[i] = Surname[i]; }
     if (!(CardDataWrite(USED_Sector,2,DataBuffer))) { CardResult = false; } //Sector 0 Block 2 Vorname mit Key A schreiben
     }
    if (Givenname.length() > 15) { a = 15; } else { a = Givenname.length(); }
    if (Givenname.length() > 0)
     {
     for (i = 0; i <= 16; i++) { DataBuffer[i] = 0; }
     for (i = 0; i <= a; i++) { DataBuffer[i] = Givenname[i]; }
     if  (!(CardDataWrite(USED_Sector,3,DataBuffer))) { CardResult = false; } //Sector 0 Block 3 Nachname mit Key A
schreiben
     }
    if (!(SetSectorAccessControl (USED_Sector,MiFareClassicKey.Key_A,MiFareClassicKey.Key_B))) { CardResult = false; }
// (byte Sector,byte Akey[6],byte Bkey[6])
    } else {
          CardResult = false;
          return CardResult;
          }
    } else {
          CardResult = false;
          return CardResult;
          }

   if (CardResult)
    {
    //Serial.println("PICC written");
    CardResult = true;
    }
   else
    {
    //Serial.println("PICC not empty");
    CardResult = false;
    }
   yield();
   return CardResult;
  }


 boolean ReadMiFareClassicPICC ()
 {
 boolean CardResult;
 byte i,a ;
 CardResult = true;
 if (CardAuthenticate(KEYA,USED_Sector,MiFareClassicKey.Key_A)) // Sector  Autenticate with READ Key A
  {
  Givenname = "aaaaaaaaaaaaaaaa"; //PlaceHolder
  Surname = "aaaaaaaaaaaaaaaa"; //PlaceHolder
  for (i = 0; i < 18; i++) { DataBuffer[i] = 0; } // Clear Variable Buffer
  if (CardDataRead(USED_Sector,2)) // Feld Vorname auslesen
   {
   for (i = 0; i < 16; i++) { Surname[i] = char(DataBuffer[i]);  }
   } else {
```

```
          return false;
          }
    for (i = 0; i < 18; i++) { DataBuffer[i] = 0; } // Clear Variable Buffer
    if (CardDataRead(USED_Sector,3)) // Feld Nachname auslesen
      {
      for (i = 0; i < 16; i++) { Givenname[i] = char(DataBuffer[i]); }
      } else {
            return false;
            }
    } else
    {
    return false;
    }
Serial.print ("ATAUTH_S:");
Serial.println (Surname);
Serial.print ("ATAUTH_G:");
Serial.println (Givenname);
return true;
}

void CardServer()
{
#define PCD_Poll_Interval 400
#define PCD_Watchdog_Interval 60000
if (millis() - PCD_ServiceCall_Handler >= PCD_Poll_Interval)
  {
   PCD_ServiceCall_Handler = millis();

    if (mfrc522.PICC_IsNewCardPresent())   // PICC = proximity integrated circuit card = kontaktlose Chipkarte
    {
     mfrc522.PICC_ReadCardSerial();
      yield();
    // Unterscheidung nach Kartentyp
    // 0x08 für MIFARE Classic 1K
    // 0x18 für MIFARE Classic 4K
    // 0x11 für MIFARE PLUS
    if (mfrc522.uid.sak == 0x08 || mfrc522.uid.sak == 0x18)
      {
      //  MiFare_Classic_Processor START (mfrc522.uid.sak);  // Nur ausführen wenn Eine Mifare Classic Karte vor den Leser
gehalten wurde.
      byte tkey[6];
      for (byte i = 0; i <= 6; i++) { tkey[i] = 255; }  //Load Default Key for all Sectors
      if(LearnNewCard) // neue Karte soll angelernt werden.
        {
          if (WriteNewMiFareClassicPICC()) { SetRGBLed(0,255,0,false); } else { SetRGBLed(255,0,0,false); }
        }

      else if (EraseCard)  // KartenDaten sollen gelöscht werden.
        {
          if (ResetCardToDefault()) { SetRGBLed(0,255,0,false); } else { SetRGBLed(255,0,0,false); }
        }
      else
        {
        if (ReadMiFareClassicPICC())
          { // Karte gültig !
          bool PinState= digitalRead(RELAIS_PIN);
          PinState = !PinState;
          digitalWrite(RELAIS_PIN, PinState);
          SetRGBLed(0,255,0,false);        //Led Grün
          } else { SetRGBLed(255,0,0,false); }
          }
      LearnNewCard = false;

      //  MiFare_Classic_Processor STOP (mfrc522.uid.sak);
      } else if (mfrc522.uid.sak == 0x00) // Mifare Ultralight
      {
        SetRGBLed(255,0,0,false);
      } else
      {
        SetRGBLed(255,0,0,false);
        //Serial.print("PICC Type not supported. Type:");
        //Serial.println(mfrc522.uid.sak); //Erweiterung: evtl andere Kartentypen
      }
    mfrc522.PCD_StopCrypto1();
    mfrc522.PICC_HaltA();
    delay(2000);
    SetRGBLed(0,0,255,false);        //Led Farbe Blau Leser ist in Grundzustand
```

```
    }
  }

if (millis() - PCD_WatchDog_Handler >= PCD_Watchdog_Interval)
  {
   PCD_WatchDog_Handler = millis();
   Result = mfrc522.PCD_PerformSelfTest();
   yield();
   mfrc522.PCD_Init();                    // Initialisiere MFRC522 Lesemodul erneut
   mfrc522.PCD_SetAntennaGain(mfrc522.RxGain_max); // Setzt Antenne auf max. Empfang
   mfrc522.PCD_AntennaOn();
   yield();
   if (!(Result))
    {
      PCDHardReset();
      Serial.println("ATPCD:ERR_H");
    }
  }
}

// ****************** Stop Functions CardServices  ************************************



void loop()  // Hauptschleife
{
  CardServer();          // Kartenleser spezifische Anfragen bearbeiten
  yield();
  server.handleClient(); // Webserveranfragen bearbeiten
  //ESP.wdtFeed();       // Watchdog zurückstellen. Deaktivieren mit "wdt_disable();"
}
```

Die WLAN Verbindungsdaten bleiben erhalten, da wir diese im vorhergehenden Sketch in einem nicht flüchtigen Speicher abgelegt haben.

Wie aber nun funktioniert jetzt aber nun die Authentifikation der Karte?

Vereinfacht gesagt, versucht das Kartenlesermodul beim Vorhalten einer Karte des Typs MiFare Classic (dies wird Vorfeld Überprüft) einen Sektor, der nur mit gültigem Schlüssel lesbar ist, zu lesen.

Ist die Aktion erfolgreich, werden die Namensfelder Vorname und Nachname ausgelesen und die Karte als gültig erkannt. (Relais wird geschaltet) Ist die Authentifizierung nicht erfolgreich oder wird der falsche Schlüssel verwendet, wird die Karte als ungültig abgewiesen.

Durch die Unterscheidung zwischen "Karte gültig oder Karte ungültig" an diesem Merkmal ergeben sich für uns folgende Vorteile:

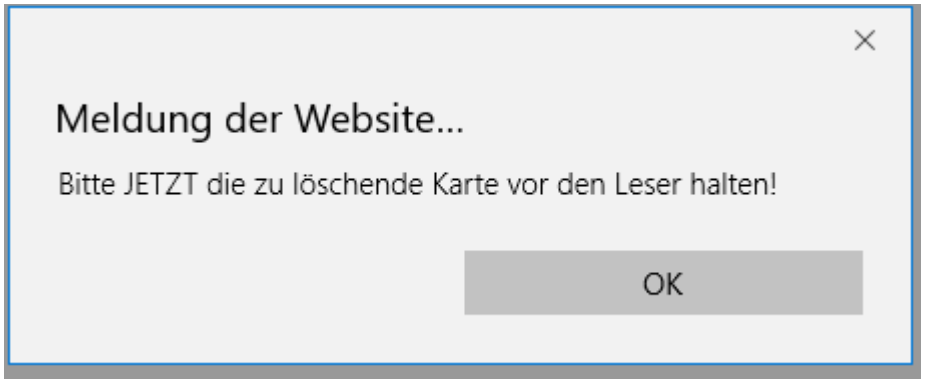
1. Es können gleichzeitig mehrere Karten an einem Leser berechtigt sein, ohne das jede Karte dem Leser bereits im Vorfeld per UUID bekannt gemacht werden muss.
2. Es können gleichzeitig mehrere Kartenleser mit einer Karte bedient werden ohne das jeder Kartenleser jede Karte bereits im Vorfeld per UUID kennt.
3. Durch Variation der Sektorendefinition oder des Schlüsselmaterials können "Berechtigungskreise" gebildet werden

Wir erstellen nun unsere erste Berechtigungskarte. Dazu gehen wir, nach Login auf dem Menüpunkt „PICC“, dann „neue Karte erstellen“ und klicken darauf. Wir sehen folgende Seite:

Wir geben ein Vor- und einen Nachnamen in die vorgesehenen Felder ein, und drücken auf den Button „Neue Smartcard erstellen“

Die LED wird gelb. Wir halten jetzt eine LEERE!! Mifare Classic Karte vor den Leser und warten kurz bis die LED grün wird. Das war’s! Ab sofort können wir diese Karte nun als autorisierte Karte an unserem Leser nutzen.

Wollen wir diese Karte ungültig machen, gehen wir, nach Login auf dem Menüpunkt PICC, dann unter „karte löschen“ und klicken darauf. -- Nach der Meldung:



halten wir die zuvor autorisierte Karte wieder vor der Leser. Diese ist danach ungültig.

Viel Spaß beim Nachbauen uns bis zum nächsten Teil.