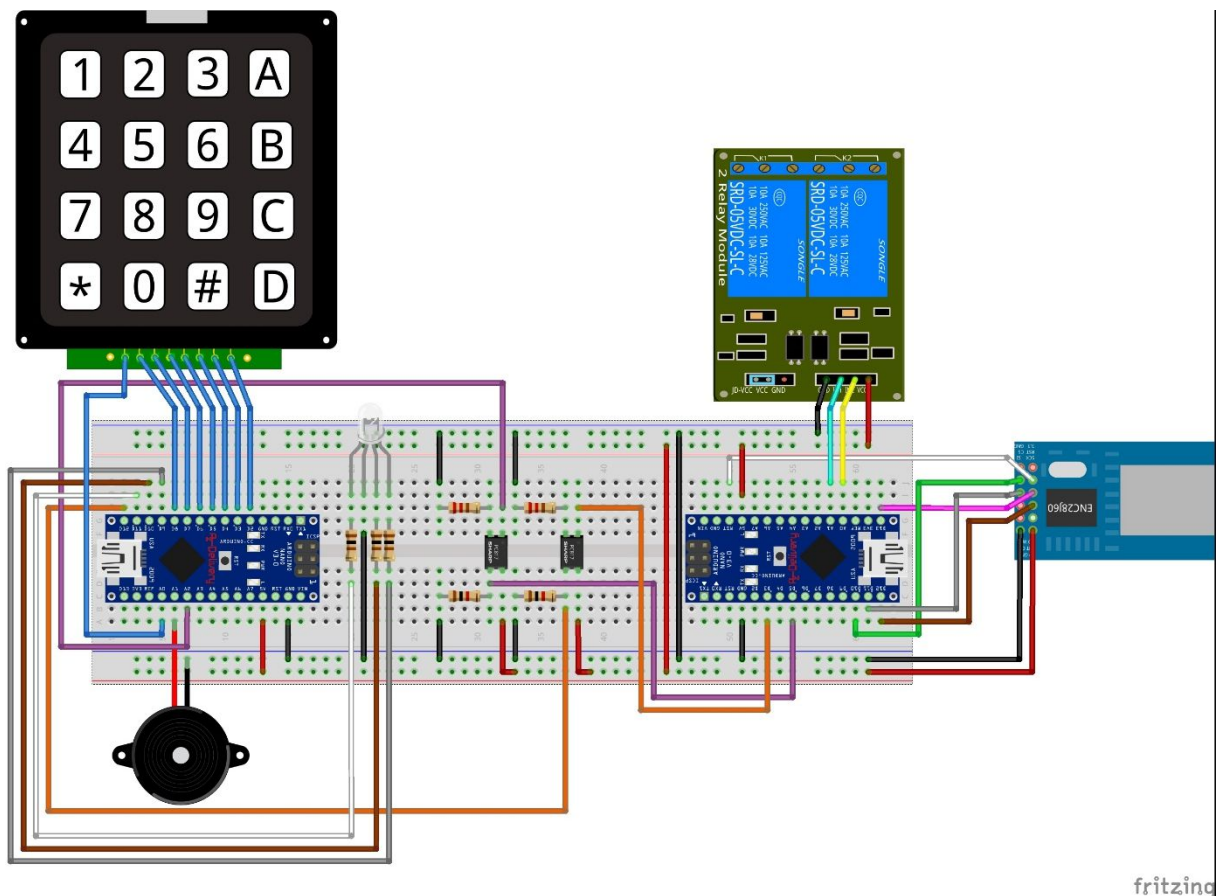


Ein letzter Feinschliff für unser Codeschloss (Teil 6)

Hallo und willkommen zu dem letzten Teil unserer Codeschlossreihe. In den letzten Teilen haben wir unser Codeschloss sowohl hardwaretechnisch als auch aus sicherheitstechnisch weiterentwickelt. Meiner Meinung nach ist das Codeschloss im Funktionsumfang bereits jetzt einigen kommerziellen Lösungen überlegen. Heute gehen wir also nicht mehr "große" Dinge bzw. Veränderungen an, sondern geben unserem Codeschloss dagegen nur noch ein wenig "Feinschliff". Da wir bereits eine funktionsfähige Netzwerkverbindung mit einer Grundfunktion der Pin-Änderung haben, bauen wir also die Netzwerkfunktionen noch ein wenig aus und kommen so zu einem komfortabel per Netzwerk steuerbaren Codeschloss. Dazu müssen wir die Hardware noch ein wenig wie in folgendem Bild gezeigt umbauen:



Hauptsächlich ändern wir in der Hardware die Eingänge der Relais im Unterschied zu den Vorgängerteilen so, das Relais 1 mit dem Port A0 und Relais 2 mit dem Port A1 verbunden wird. Damit wird Relais 1 nun schaltbar per Code (Pin) UND per Netzwerk darüber hinaus ist nun noch Relais 2 schaltbar, allerdings ausschließlich per Netzwerk. Als Hardware für die Netzwerkanbindung kann, wie im vorherigen Teil auch, folgende Module verwendet werden:

- 1.) [ENC28J60 Ethernet Shield für Arduino Nano V3.0](#)
- 2.) [ENC28J60 Ethernet Shield LAN Netzwerk Modul für Arduino](#)

Ich empfehle auch weiterhin die Verwendung des ENC28J60 Ethernet Shield für Arduino Nano in Version 3.0, da dieses Modul einen recht einfachen und platzsparenden Aufbau ermöglicht.

Wir brauchen für den letzten Teil der Reihe folgende Hardware:

Anzahl	Bezeichnung	Anmerkung
1	Relais Modul	
2	Arduino Nano	
1	4x4 Keypad	
3	Widerstände 120 Ohm	
1	RGB Led	
2	Optokoppler Sharp PC817	
2	Widerstände 220 Ohm	Strombeg. Eing. Optokoppler
2	Widerstände 1 KOhm	Strombeg. Ausg. Optokoppler
1	ENC28J60 Ethernet f. A. Nano	
1	ENC28J60 Ethernet f. Arduino	Alternative

Für die neuen Funktionen bauen wir den Befehlssatz, den das Codeschloss akzeptiert aus. Nachfolgend möchte ich daher vorab die NEU hingekommenen Befehle und Funktionen vorstellen bzw. beschreiben:

Befehl:

CodeB: (maximal 20 Zeichen aus dem Zeichensatz 0-9 und A-D werden akzeptiert.)

Erklärung:

Dient zum Festlegen eines zweiten ggf. gültigen Codes, der ZUSÄTZLICH als gültig akzeptiert werden kann. Beispielanwendung: Zweiter berechtigter Anwender der das Codeschloss bedienen können soll, aber ein eigener Code haben soll.

Befehl:

codeb:enabled/disabled

Erklärung:

Dienst zum Aktivieren oder deaktivieren des zweiten konfigurierbaren Codes, der bei enabled als gültig akzeptiert wird. Falls dieser per Parameter disabled deaktiviert wird, wird der dazugehörige Code NICHT gelöscht und kann jederzeit reaktiviert werden.

Befehl:

codea:enabled/disabled

Erklärung:

Dienst zum Aktivieren oder deaktivieren des ersten konfigurierbaren Codes, der bei enabled als gültig akzeptiert wird. Falls dieser per parameter disabled deaktiviert wird, wird der dazugehörige Code NICHT gelöscht und kann jederzeit reaktiviert werden.

Befehl:

lock:enabled/disabled

Erklärung:

Dienst zum deaktivieren oder (re)aktivieren des Codeschlosses als Ganzes. Nach Absetzen des Befehls lock:disabled wird eine weitere Eingabe komplett gesperrt, die LED auf der Anzeigeeinheit leuchtet dauerhaft rot. Der Befehl um dies rückgängig zu machen lautet lock:enabled

Befehl:

toggle:a

Erklärung:

Dient zur Betätigung des Coderelais per Netzwerk. Dies ist gleichzusetzen mit einer korrekten Pineingabe ! Auf der Eingabeeinheit leuchtet kurz die grüne LED auf und es wird der "Code korrekt" Ton ausgegeben.

Befehl:

toggle:b

Erklärung:

Dient zur Betätigung des Zusatzrelais per Netzwerk. D Auf der Eingabeeinheit wird dies nicht quittiert. Dieses Relais kann ausschließlich über diesen Weg gesteuert werden.

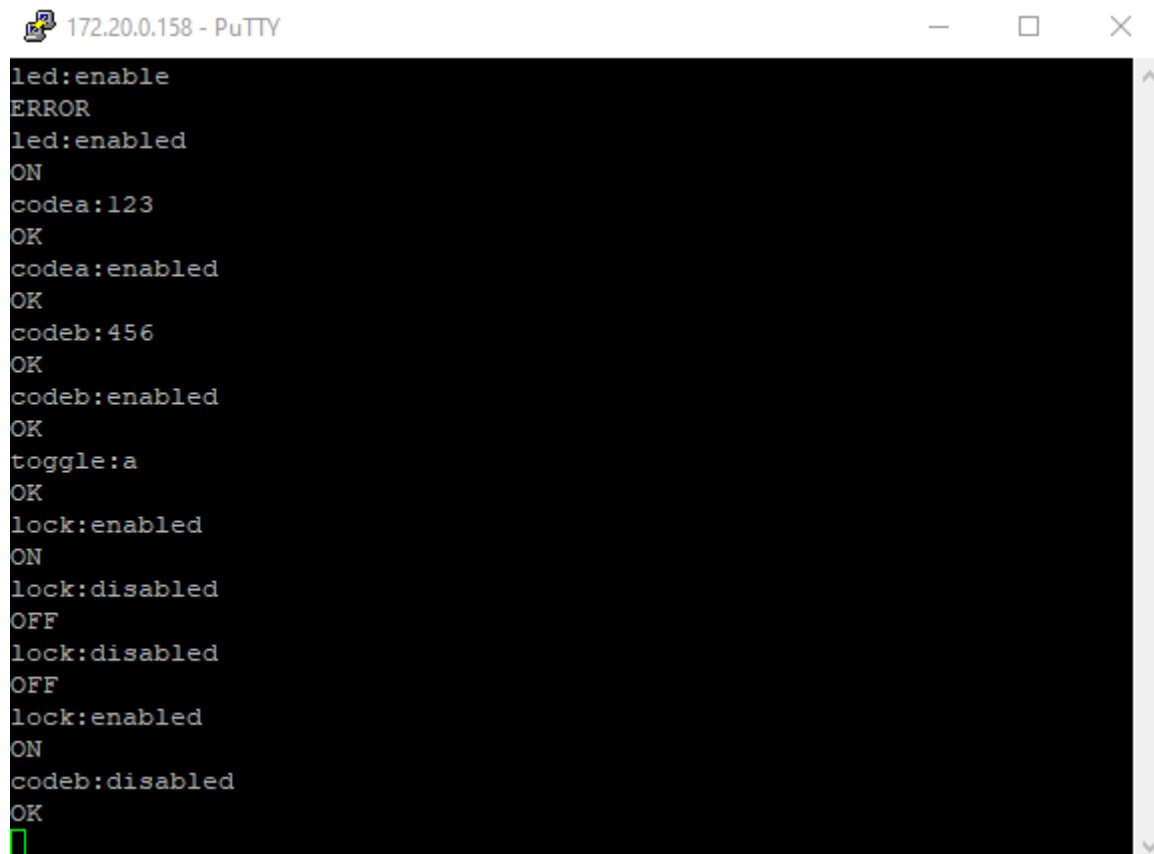
Befehl:

led:disable/enable

Erklärung:

Mit diesem Befehl led:disable wird das dauerleuchten der LED in den farben blau (Bereitschaftsmodus) magenta (Anzeige Fehleingabe des Codes und Bereitschaft) und cyan (Es läuft eine aktuelle Eingabe) unterdrückt. Es wird nur der Status unsynchronisiert (gelb), warten auf Synchronisierung (weiß) und Code Falsch (rot) und Code richtig (grün) ausgegeben. Wenn das Codeschloss in einem Bereich untergebracht ist, wo das permanente Leuchten stört, kann dies so abgeschaltet werde. Achtung: Status ist nicht gegen Stromausfall gesichert! Der Gegenbefehl lautet: led:enable

Alle (neue) Befehle können per Netzwerk über das Protokoll Telnet auf Port 22 auf dem Codeschloss genutzt werden. Ein nützliches Telnet Programm ist das Programm „Putty“ das auf der Webseite <https://www.putty.org/> heruntergeladen werden kann.



```
172.20.0.158 - PuTTY
led:enable
ERROR
led:enabled
ON
codea:123
OK
codea:enabled
OK
codeb:456
OK
codeb:enabled
OK
toggle:a
OK
lock:enabled
ON
lock:disable
OFF
lock:disable
OFF
lock:enabled
ON
codeb:disable
OK
█
```

Bitte beachten:

Durch umfassenden Steuerungsmöglichkeiten per Netzwerk ist das Netzwerk, in dem das Codeschloss hängt, unbedingt gegen die Zugriffe von Unbefugten zu sichern. Jeder der Zugriff auf das Netzwerk hat kann o.g. Befehle absetzen. Es wird empfohlen, das Netzwerk nicht per WLAN zugänglich zu machen.

Um die neuen beschriebenen Funktionen nutzen zu können, laden Sie bitte folgenden aktualisierten Code auf das Eingabeteil hoch:

```
// Codeschloss Tobias Kuch 2020 GPL 3.0
#include <Keypad.h>
#include <SoftwareSerial.h>

#define RGBLED_R 11
#define RGBLED_G 10
#define RGBLED_B 9
#define RGBFadeInterval1 10 // in ms
#define KeybModeTimeInterval1 5000 // in ms
#define PIEZOSUMMER A1
#define CyclesInBlackMax 20

#define RGBOFF 0
#define RGBSHORTBLACK 8
#define RGBRED 1
#define RGBGREEN 2
#define RGBBLUE 3
#define RGBWHITE 4
#define RGBYELLOW 5
#define RGBCYAN 6
#define RGBMAGENTA 7

const byte ROWS = 4;
const byte COLS = 4;
const byte MaxPinCodeLength = 20;

SoftwareSerial mySerial(12, A2); // RX, TX

char keys[ROWS][COLS] = {
    {49,50,51,65},
    {52,53,54,66},
    {55,56,57,67},
    {58,48,59,68},
};

byte colPins[COLS] = {A0,8,7,6}; //A0,8,7,6;
byte rowPins[ROWS]= {5,4,3,2}; // 5,4,3,2}

byte RGBValue_R = 0;
byte RGBValue_G = 0;
byte RGBValue_B = 0;
byte RGBFadeValue_R = 0;
byte RGBFadeValue_G = 0;
byte RGBFadeValue_B = 0;
bool RGBFadeDir_R = true;
bool RGBFadeDir_G = true;
```

```

bool RGBFadeDir_B = true;

byte key = 0;
bool InSync = true;
bool CodeEnterSequence = false;
bool CodeEnterSequenceOLD = false;
bool InputBlocked = false;
bool PinEnteredFalseBefore = false;
bool RGBFadeEnabled = true;
bool DisplayStatusLed = true;

long previousMillis = 0;
long previousMillisKeyBoard = 0;
byte EnCodedKeyStroke = 0;
byte inByte = 0;
int CyclesInBlack = 0;
byte ReclnititalKeyLength = 0;
unsigned long InititalKey = 0;

Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

union foo {
    byte as_array[4];
    long as_long;
} d;

void setup()
{
    mySerial.begin(9600);
    Serial.begin(9600);
    pinMode(RGBLED_G,OUTPUT); // Ausgang RGB LED Grün
    pinMode(RGBLED_R,OUTPUT); // Ausgang RGB LED Rot
    pinMode(RGBLED_B,OUTPUT); // Ausgang RGB LED Blau
    pinMode(PIEZOSUMMER,OUTPUT); // Ausgang RGB LED Blau
    digitalWrite(PIEZOSUMMER,LOW); // Ausgang RGB LED Blau
    RGBControl(RGBWHITE,false); // INIT MODE
    ReclnititalKeyLength = 0;
    do
    {
        if (mySerial.available())
        {
            inByte = mySerial.read();
            d.as_array[ReclnititalKeyLength]=inByte; //little Endian
            ReclnititalKeyLength++;
        }
    } while (ReclnititalKeyLength < 4);
    InititalKey = d.as_long;
    randomSeed(InititalKey);

```

```
RGBControl(RGBBLUE,true); // NORMAL MODE  
}
```

```
void RGBControl(byte function, bool fadeit)
```

```
{  
  if (function == RGBOFF)  
  {  
    RGBValue_R = 0;  
    RGBValue_G = 0;  
    RGBValue_B = 0;  
    RGBFadeValue_R = 0;  
    RGBFadeValue_G = 0;  
    RGBFadeValue_B = 0;  
    RGBFadeDir_R = true;  
    RGBFadeDir_G = true;  
    RGBFadeDir_B = true;  
  }  
  if (function == RGBRED)  
  {  
    RGBValue_R = 255;  
    RGBValue_G = 0;  
    RGBValue_B = 0;  
    RGBFadeValue_R = 255;  
    RGBFadeValue_G = 0;  
    RGBFadeValue_B = 0;  
    RGBFadeDir_R = false;  
    RGBFadeDir_G = true;  
    RGBFadeDir_B = true;  
  }  
  if (function == RGBGREEN)  
  {  
    RGBValue_R = 0;  
    RGBValue_G = 255;  
    RGBValue_B = 0;  
    RGBFadeValue_R = 0;  
    RGBFadeValue_G = 255;  
    RGBFadeValue_B = 0;  
    RGBFadeDir_R = true;  
    RGBFadeDir_G = false;  
    RGBFadeDir_B = true;  
  }  
  if (function == RGBBLUE)  
  {  
    if (DisplayStatusLed)  
    {  
      RGBValue_R = 0;  
      RGBValue_G = 0;  
      RGBValue_B = 255;  
      RGBFadeValue_R = 0;  
      RGBFadeValue_G = 0;  
      RGBFadeValue_B = 255;  
    }  
  }  
}
```

```

    RGBFadeDir_R = true;
    RGBFadeDir_G = true;
    RGBFadeDir_B = false;
} else
{ // LED OFF
    fadeit = false;
    RGBValue_R = 0;
    RGBValue_G = 0;
    RGBValue_B = 0;
    RGBFadeValue_R = 0;
    RGBFadeValue_G = 0;
    RGBFadeValue_B = 0;
    RGBFadeDir_R = true;
    RGBFadeDir_G = true;
    RGBFadeDir_B = true;
}
}
if (function == RGBWHITE)
{
    RGBValue_R = 255;
    RGBValue_G = 255;
    RGBValue_B = 255;
    RGBFadeValue_R = 255;
    RGBFadeValue_G = 255;
    RGBFadeValue_B = 255;
    RGBFadeDir_R = false;
    RGBFadeDir_G = false;
    RGBFadeDir_B = false;
}
if (function == RGBCYAN)
{
    if (DisplayStatusLed)
    {
        RGBValue_R = 0;
        RGBValue_G = 255;
        RGBValue_B = 255;
        RGBFadeValue_R = 0;
        RGBFadeValue_G = 255;
        RGBFadeValue_B = 255;
        RGBFadeDir_R = true;
        RGBFadeDir_G = false;
        RGBFadeDir_B = false;
    } else
    { // LED OFF
        fadeit = false;
        RGBValue_R = 0;
        RGBValue_G = 0;
        RGBValue_B = 0;
        RGBFadeValue_R = 0;
        RGBFadeValue_G = 0;
        RGBFadeValue_B = 0;
    }
}

```



```

    RGBFadeDir_R = true;
    RGBFadeDir_G = true;
    RGBFadeDir_B = true;
}
}
if (function == RGBYELLOW)
{
    RGBValue_R = 255;
    RGBValue_G = 255;
    RGBValue_B = 0;
    RGBFadeValue_R = 0;
    RGBFadeValue_G = 0;
    RGBFadeValue_B = 0;
    RGBFadeDir_R = true;
    RGBFadeDir_G = true;
    RGBFadeDir_B = true;
}
if (function == RGBMAGENTA)
{
    if (DisplayStatusLed)
    {
        RGBValue_R = 255;
        RGBValue_G = 0;
        RGBValue_B = 255;
        RGBFadeValue_R = 255;
        RGBFadeValue_G = 0;
        RGBFadeValue_B = 255;
        RGBFadeDir_R = false;
        RGBFadeDir_G = true;
        RGBFadeDir_B = false;
    } else
    { // LED OFF
        fadeit = false;
        RGBValue_R = 0;
        RGBValue_G = 0;
        RGBValue_B = 0;
        RGBFadeValue_R = 0;
        RGBFadeValue_G = 0;
        RGBFadeValue_B = 0;
        RGBFadeDir_R = true;
        RGBFadeDir_G = true;
        RGBFadeDir_B = true;
    }
}
if (function == RGBSHORTBLACK)
{
    analogWrite(RGBLED_R, 0);
    analogWrite(RGBLED_G, 0);
    analogWrite(RGBLED_B, 0);
}
}
RGBFadeEnabled = fadeit;

```

```

if (!(RGBFadeEnabled))
{
    analogWrite(RGBLED_R, RGBValue_R);
    analogWrite(RGBLED_G, RGBValue_G);
    analogWrite(RGBLED_B, RGBValue_B);
}
}

void SerialHandler ()
{

if (mySerial.available())
{
    inByte = mySerial.read();
    if (inByte == 30) // Eingabe gesperrt Zeitschloss aktiv
    {
        InputBlocked = true;
        RGBControl(RGBRED,true);
    }
    if (inByte == 40) // Eingabe entsperrt Zeitschloss deaktiviert
    {
        RGBControl(RGBMAGENTA,true);
        InputBlocked = false;
        tone(PIEZOSUMMER, 880, 100);
        delay(120);
    }
    if (inByte == 41) // Eingabe entsperrt Zeitschloss deaktiviert, normaler Modus
    {
        RGBControl(RGBBLUE,true);
        InputBlocked = false;
        tone(PIEZOSUMMER, 880, 100);
        delay(120);
    }
    if (inByte == 20) // Code Correct
    {
        RGBControl(RGBGREEN,false);
        tone(PIEZOSUMMER, 1200, 200);
        delay(2000);
        PinEnteredFalseBefore = false;
        RGBControl(RGBBLUE,true); // NORMAL MODE
    }
    if (inByte == 21) // Code falsch
    {
        analogWrite(RGBLED_R, 255);
        analogWrite(RGBLED_G, 0);
        analogWrite(RGBLED_B, 0);
        tone(PIEZOSUMMER, 400, 300);
        delay(500);
        RGBControl(RGBRED,true);
        InputBlocked = true;
    }
}
}

```

```

    PinEnteredFalseBefore = true;
}
if (inByte == 25) // Out of Sync
{
    RGBControl(RGBYELLOW,true);
    InSync = false;
    InititalKey = 0; // Delete Encryption Key
}
if (inByte == 23) //Clear ausgeführt
{
    inByte = 0;
}
if (inByte == 22) // Eingabe azeptiert
{
    inByte = 0;
}
if (inByte == 27) // Eingabe azeptiert
{
    DisplayStatusLed = true;
    if (PinEnteredFalseBefore)
    {
        RGBControl(RGBMAGENTA,true);
    } else
    {
        RGBControl(RGBBLUE,true); // NORMAL MODE
    }
    inByte = 0;
}
if (inByte == 26) // Eingabe azeptiert
{
    DisplayStatusLed = false;
    analogWrite(RGBLED_R, 0);
    analogWrite(RGBLED_G, 0);
    analogWrite(RGBLED_B, 0);
    RGBControl(RGBOFF,false);
    inByte = 0;
}
}
}

void TimeMgmnt ()
{
    if ((millis() - previousMillisKeyBoard > KeybModeTimeInterval1) &
    CodeEnterSequence & InSync) // Auto Reset KEYboard Input
    {
        previousMillisKeyBoard = millis();
        tone(PIEZOSUMMER, 988, 100);
        delay(110);
        if (PinEnteredFalseBefore)
        {

```

```

    RGBControl(RGBMAGENTA,true); // NORMAL MODE - Pin entered false before
  } else
  {
    RGBControl(RGBBLUE,true); // NORMAL MODE
  }
  CodeEnterSequence = false;
  previousMillisKeyBoard = millis();
  byte randNumber = random(0, 254);
  EnCodedKeyStroke = 58 ^ randNumber;
  mySerial.write(EnCodedKeyStroke);
}
if (millis() - previousMillis > RGBFadeInterval1) //Fadint LEd's
{
  if (RGBFadeEnabled)
  {
    previousMillis = millis(); // aktuelle Zeit abspeichern
    if (RGBValue_B > 0)
    {
      if (RGBFadeDir_B)
      {
        RGBFadeValue_B++;
        if ( RGBFadeValue_B >= RGBValue_B) {RGBFadeDir_B = false; }
      } else
      {
        RGBFadeValue_B--;
        if ( RGBFadeValue_B < 1) {RGBFadeDir_B = true; }
      }
    } else { RGBFadeValue_B = 0; }
    if (RGBValue_R > 0)
    {
      if (RGBFadeDir_R)
      {
        RGBFadeValue_R++;
        if ( RGBFadeValue_R >= RGBValue_R) {RGBFadeDir_R = false; }
      } else
      {
        RGBFadeValue_R--;
        if ( RGBFadeValue_R < 1) {RGBFadeDir_R = true; }
      }
    } else { RGBFadeValue_R = 0; }
    if (RGBValue_G > 0)
    {
      if (RGBFadeDir_G)
      {
        RGBFadeValue_G++;
        if ( RGBFadeValue_G >= RGBValue_G) {RGBFadeDir_G = false; }
      } else
      {
        RGBFadeValue_G--;
        if ( RGBFadeValue_G < 1) {RGBFadeDir_G = true; }
      }
    }
  }
}

```

```

    } else { RGBFadeValue_G = 0; }
    analogWrite(RGBLED_R, RGBFadeValue_R);
    analogWrite(RGBLED_G, RGBFadeValue_G);
    analogWrite(RGBLED_B, RGBFadeValue_B);
  }
}
}

void KeyboardHandler(bool NotEnabled)
{
  key = keypad.getKey();
  if((key)) // Key Entered
  {
    if (!NotEnabled)
    {
      byte randNumber = random(0, 254);
      EnCodedKeyStroke = key ^ randNumber;
      mySerial.write(EnCodedKeyStroke);
      if((key == 58) | (key == 59))
      {
        RGBControl(RGBSHORTBLACK,true);
        tone(PIEZOSUMMER, 988, 100);
        delay(120);
        CodeEnterSequence = false;
        if(key == 58)
        {
          if (PinEnteredFalseBefore)
          {
            RGBControl(RGBMAGENTA,true); // NORMAL MODE - Pin entered false
before
          } else
          {
            RGBControl(RGBBLUE,true); // NORMAL MODE
          }
        }
      } else
      {
        RGBControl(RGBSHORTBLACK,true);
        tone(PIEZOSUMMER, 880, 100);
        delay(120);
        CodeEnterSequence = true;
        RGBControl(RGBCYAN,true);
        previousMillisKeyBoard = millis();
      }
    }
  }
}

void loop()
{

```

```

if (InSync)
{
    KeyboardHandler(InputBlocked);
}
TimeMgmnt ();
SerialHandler ();
}

```

Anschließend laden Sie bitte folgenden aktualisierten Code auf die Auswerteeinheit hoch:

```

#include <SoftwareSerial.h>
#include <UIPEthernet.h>
#include <EEPROM.h>

#define RELAIS_A A0
#define RELAIS_B A1
#define MACADDRESS 0x00,0x01,0x02,0x03,0x04,0x05
#define MYIPADDR 192,168,1,6
#define MYIPMASK 255,255,255,0
#define MYDNS 192,168,1,1
#define MYGW 192,168,1,1
#define LISTENPORT 23
#define UARTBAUD 9600

#define Interval1 1000
#define MAXDelayStages 7

const byte MaxPinCodeLength = 20;
const byte DelayIterationsInSec[MAXDelayStages] = {1,5,10,20,30,45,60};

SoftwareSerial mySerial(5, 3); // RX, TX
EthernetServer server = EthernetServer(LISTENPORT);
EthernetClient client;
EthernetClient newClient;

byte KeyPadBuffer[MaxPinCodeLength];
byte BufferCount = 0;
byte a;
bool InSync = true;
byte ErrorCounter = 0;
long previousMillis = 0;
byte InputDelay = 0;
byte ReclnitialKeyLength = 0;
uint8_t mac[6] = {MACADDRESS};
unsigned long CommuncationKey = 902841;

union foo {
    byte as_array[4];
    long as_long;
} convert;

```

```

struct EEPROMData
{
    byte PinCodeA[MaxPinCodeLength];
    bool PinCodeAEnabled = false;
    byte PinCodeB[MaxPinCodeLength];
    bool PinCodeBEnabled = false;
    char ConfigValid[3]; //If Config is Vaild, Tag "TK" is required"
};

EEPROMData MyPinCodes;

void SaveCodesToEEPROM ()
{
    for (int i = 0 ; i < sizeof(MyPinCodes) ; i++)
    {
        EEPROM.write(i, 0);
    }
    strncpy(MyPinCodes.ConfigValid , "TK", sizeof(MyPinCodes.ConfigValid) );
    EEPROM.put(0, MyPinCodes);
}

bool GetCodesFromEEPROM ()
{
    bool RetValue;
    EEPROM.get(0, MyPinCodes);
    if (String(MyPinCodes.ConfigValid) == String("TK"))
    {
        RetValue = true;
    } else
    {
        RetValue = false;
    }
    return RetValue;
}

void setup()
{
    Serial.begin(9600);
    mySerial.begin(9600);
    pinMode(RELAIS_A,OUTPUT); //Relais Output
    pinMode(RELAIS_B,OUTPUT); //Relais Output
    digitalWrite(RELAIS_A,HIGH); //LOW Aktiv
    digitalWrite(RELAIS_B,HIGH); //LOW Aktiv
    if (!(GetCodesFromEEPROM()))
    {
        Serial.println (F("Empty EEPROM."));
        MyPinCodes.PinCodeA[0] = 49; // Default Pincode: 123
        MyPinCodes.PinCodeA[1] = 50;
        MyPinCodes.PinCodeA[2] = 51;
        MyPinCodes.PinCodeA[3] = 0;
        MyPinCodes.PinCodeAEnabled = true;
    }
}

```

```

    MyPinCodes.PinCodeAEnabled = false;
    SaveCodesToEEPROM();
}
BufferCount = 0;
for (a = 0; a <= MaxPinCodeLength - 1 ; a++)
{
    KeyPadBuffer[a] = 0;
}
convert.as_long = CommuncationKey;
ReclnitalKeyLength = 0;
Ethernet.begin(mac);
server.begin();
Serial.println(Ethernet.localIP());
do
{
    mySerial.write(convert.as_array[ReclnitalKeyLength]); //little Endian
    ReclnitalKeyLength++;
} while (ReclnitalKeyLength < 4);
randomSeed(CommuncationKey);
}

```

```

bool CheckEnabled (String s)
{
    if (s == "ENABLED")
    {
        return true;
    } else
    {
        return false;
    }
}

```

```

bool CheckDisabled (String s)
{
    if (s == "DISABLED")
    {
        return true;
    } else
    {
        return false;
    }
}

```

```

void loop()
{
    if (client = server.available())
    {
        byte Position = 0;
        bool CommandReceived = false;
        char EtherNetCommand[MaxPinCodeLength + 7] = "";
        while((client.available()) > 0)

```



```

{
  byte thisChar = client.read();
  if ((thisChar < 123) & (thisChar > 47) & (Position < MaxPinCodeLength + 6) ) // Sonderzeichen
  ausfiltern
  {
    EtherNetCommand[Position] = thisChar;
    Position++;
    if (Position > 1) { CommandReceived = true; }
  }
}
if (CommandReceived)
{
  String s(EtherNetCommand);
  s.toUpperCase();
  EtherNetCommand[MaxPinCodeLength + 7] = "";
  if (s.startsWith("TOGGLE:"))
  {
    s.remove(0, 7);
    if (s == "A")
    {
      digitalWrite(RELAIS_A,!digitalRead(RELAIS_A));
      client.println(F("OK"));
      mySerial.write(20);
    } else
    if (s == "B")
    {
      digitalWrite(RELAIS_B,!digitalRead(RELAIS_B));
      client.println(F("OK"));
    }

  } else
  if (s.startsWith("LOCK:"))
  {
    s.remove(0, 5);
    if (CheckEnabled(s))
    {
      client.println(F("OK"));
      mySerial.write(41);
    } else
    if (CheckDisabled(s))
    {
      client.println(F("OK"));
      mySerial.write(21);
    } else { client.println(F("ERROR")); }
  } else
  if (s.startsWith("LED:"))
  {
    s.remove(0, 4);
    if (CheckEnabled(s))
    {
      mySerial.write(27);
      client.println(F("OK"));
    } else

```

```

if (CheckDisabled(s))
{
    mySerial.write(26);
    client.println(F("OK"));
} else { client.println(F("ERROR")); }
} else
if (s.startsWith("CODEA:"))
{
    s.remove(0, 6);
    if (CheckEnabled(s)) //CheckEnabled // old: if (s.startsWith("ENABLED"))
    {
        MyPinCodes.PinCodeAEnabled = true;
        client.println(F("OK"));
        SaveCodesToEEPROM ();
    } else
    if (CheckDisabled(s)) // if (s.startsWith("DISABLED"))
    {
        MyPinCodes.PinCodeAEnabled = false;
        client.println(F("OK"));
        SaveCodesToEEPROM ();
    } else
    {
        Serial.println(F("Neuer Code A"));
        byte a = s.length();
        bool CodeOk = true;
        for (Position = 0; Position < a; Position++) // Check auf gültige Zeichen
        {
            if ((char(s[Position])<48) | (char(s[Position])>68) | ((char(s[Position])>57) &
(char(s[Position])<65))) { CodeOk = false; }
        }
        if (CodeOk)
        {
            client.println(F("OK"));
            Serial.println(s);
            for (Position = 0; Position < MaxPinCodeLength; Position++) // Check auf gültige Zeichen
            {
                MyPinCodes.PinCodeA[Position] = 0;
            }
            for (Position = 0; Position < a; Position++) // Check auf gültige Zeichen
            {
                MyPinCodes.PinCodeA[Position] = s[Position];
            }
            SaveCodesToEEPROM ();
        } else { client.println(F("ERROR")); }
    }
} else
if (s.startsWith("CODEB:"))
{
    s.remove(0, 6);
    if (CheckEnabled(s))
    {
        MyPinCodes.PinCodeBEnabled = true;
        client.println(F("OK"));
    }
}

```

```

        SaveCodesToEEPROM ();
    } else
    if (CheckDisabled(s))
    {
        MyPinCodes.PinCodeBEnabled = false;
        client.println(F("OK"));
        SaveCodesToEEPROM ();
    } else
    {
        Serial.println(F("Neuer Code B"));
        byte a = s.length();
        bool CodeOk = true;
        for (Position = 0; Position < a; Position++) // Check auf gültige Zeichen
        {
            if ((char(s[Position])<48) | (char(s[Position])>68) | ((char(s[Position])>57) &
(char(s[Position])<65))) { CodeOk = false; }
        }
        if (CodeOk)
        {
            client.println(F("OK"));
            Serial.println(s);
            for (Position = 0; Position < MaxPinCodeLength; Position++) // Check auf gültige Zeichen
            {
                MyPinCodes.PinCodeB[Position] = 0;
            }
            for (Position = 0; Position < a; Position++) // Check auf gültige Zeichen
            {
                MyPinCodes.PinCodeB[Position] = s[Position];
            }
            SaveCodesToEEPROM();
        } else { client.println(F("ERROR")); }
    }
} else { client.println(F("ERROR")); }
}
}

if (client && !client.connected())
{
    Serial.print(F("Client Disconnected"));
    client.stop();
}

if (mySerial.available())
{
    byte randNumber = random(0, 254);
    byte key = mySerial.read();
    byte DeCodedKeyStroke = key ^ randNumber;
    if (((DeCodedKeyStroke > 47) & (DeCodedKeyStroke < 69)) & InSync)
    {
        if(DeCodedKeyStroke == 58) // Clear Keypad Buffer Key: *
        {
            for (a = 0; a <= MaxPinCodeLength -1; a++)
            {
                KeyPadBuffer[a] = 0;
            }
        }
    }
}

```

```

    }
    Serial.print(F("Clear "));
    // Serial.println(BufferCount);
    mySerial.write(23);
    BufferCount = 0;
} else
if(DeCodedKeyStroke ==59) // Enter Keypad Buffer Key: #
{
    if (InputDelay == 0)
    {
        //Serial.println("Auswertung gestartet"); // Zu Debugzwecken
        // Serial.println(BufferCount);
        bool AcceptCodeA = true;
        bool AcceptCodeB = true;
        if (MyPinCodes.PinCodeAEnabled)
        {
            for (a = 0; a <= MaxPinCodeLength -1 ; a++)
            {
                if (!(MyPinCodes.PinCodeA[a] == KeyPadBuffer[a])) {AcceptCodeA = false; }
                //Serial.print(MyPinCodes.PinCodeA[a]); // Zu Debugzwecken
                //Serial.print(";");
                //Serial.print(KeyPadBuffer[a]);
                // Serial.println(" ");
            }
        } else {AcceptCodeA = false; }
        if (MyPinCodes.PinCodeBEnabled)
        {
            for (a = 0; a <= MaxPinCodeLength -1 ; a++)
            {
                if (!(MyPinCodes.PinCodeB[a] == KeyPadBuffer[a])) {AcceptCodeB = false; }
            }
        } else {AcceptCodeB = false; }
        // Serial.println("END"); // Zu Debugzwecken
        if (AcceptCodeA | AcceptCodeB)
        {
            mySerial.write(20);
            digitalWrite(RELAIS_A,!digitalRead(RELAIS_A));
            ErrorCounter = 0;
            InputDelay = 0;
            AcceptCodeA = false;
            AcceptCodeB = false;
        } else
        {
            mySerial.write(21);
            if ( ErrorCounter < MAXDelayStages - 1) { ErrorCounter++; }
            InputDelay = DelayIterationsInSec [ErrorCounter];
        }
        for (a = 0; a <= MaxPinCodeLength -1; a++) { KeyPadBuffer[a] = 0; }
        Serial.println(F("Clearing Memory"));
        BufferCount = 0;
    } else
    {
        Serial.println(F("Delay Mode Active"));
    }
}

```

```

        mySerial.write(30); // Delay Mode
        for (a = 0; a <= MaxPinCodeLength - 1 ; a++) { KeyPadBuffer[a] = 0; }
        BufferCount = 0;
    }
} else
{
    KeyPadBuffer[BufferCount] = DeCodedKeyStroke;
    if (BufferCount < MaxPinCodeLength ) { BufferCount++; }
    if (InputDelay == 0) { mySerial.write(22); } else { mySerial.write(30); }
}
} else
{
    //Out of Sync
    Serial.print(F("Out of sync Data: "));
    Serial.println(DeCodedKeyStroke);
    mySerial.write(25);
    if ( ErrorCounter < MAXDelayStages - 1) { ErrorCounter++; }
    InSync = false;
}
}

if (millis() - previousMillis > Interval1)
{
    // Auto Reset KEYboard Input
    ;
    previousMillis = millis();
    if (InputDelay > 0)
    {
        if (InputDelay == 1)
        {
            Serial.println (F("Release"));
            mySerial.write(40); // Delay Mode End
        }
        InputDelay = InputDelay - 1;
    }
}
}
}

```

Achtung!

Sollten Sie kein Netzwerk während des Einschaltens des Codeschlosses angeschlossen haben, kann es etwas mehr als eine Minute dauern, bis das Codeschloss einsatzbereit ist, da in diesem Falle aufgrund einer Bibliothek internen Timeout Funktion die Begin Funktion der „UIPEthernet“ Bibliothek solange wartet, bis das Programm fortgesetzt wird. Um dies zu umgehen, kann auch eine feste IP-Adresse für das Codeschloss vergeben werden.

Ich schließe die Codeschlossreihe mit einer Anregung zur Weiterentwicklung an Sie: Zum Beispiel ist es noch denkbar, in die Eingabeeinheit ein Sabotageschutz in Form eines Schalters anzubringen, der bei Öffnung des Gehäuses auslöst, und den im

RAM gespeicherten Zugangsschlüssel löscht. Ich freue mich wie immer über Feedback.