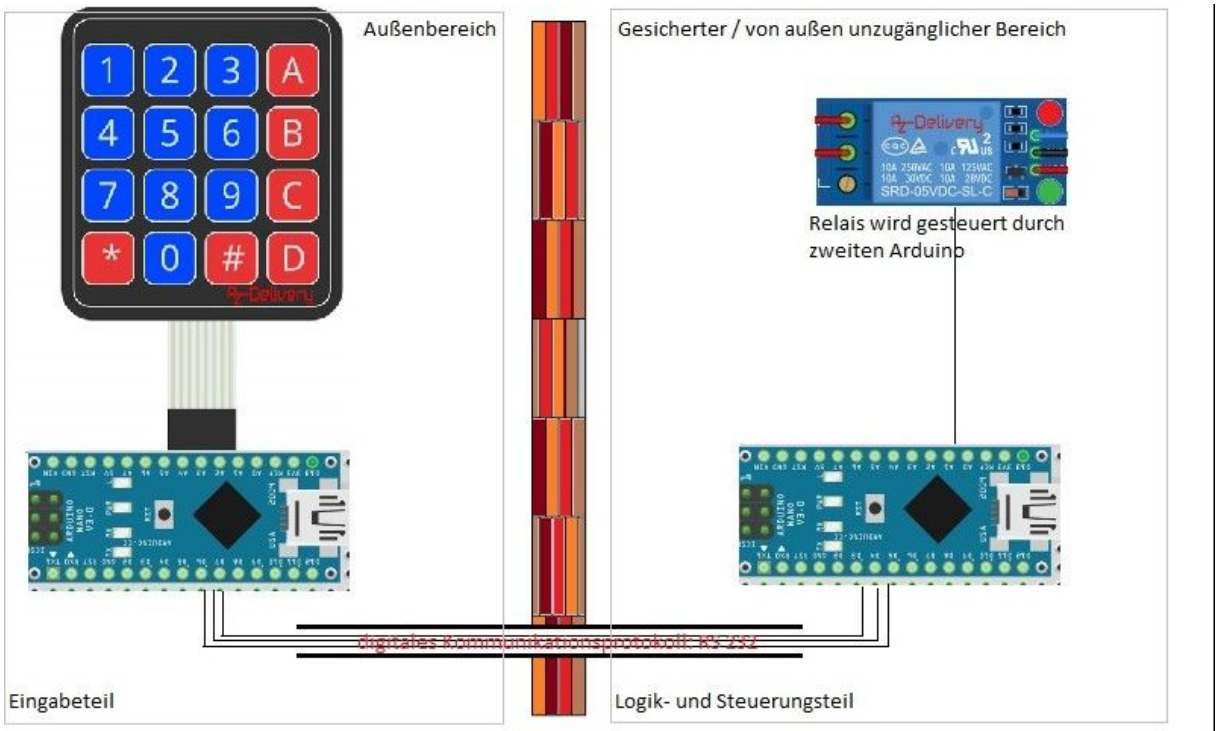
Verschlüsselte Multiprozessorkommunikation für unser Codeschloss (Teil 3)

hallo und willkommen zu einem neuen Teil unserer Codeschlossreihe. Im vorherigen Teil dieser Reihe hatte ich ja schon erwähnt, dass unser Codeschloss noch ein paar Schwachstellen besitzt. Zwar kann das Verbraucherrelais, bedingt durch eine bauliche Trennung des Bedienteiles und des Auswerteteils



nicht mehr direkt mit einfachsten Mitteln angesteuert werden, da die Tastendrücke per RS232 Schnittstellen das "Backend" weitergeleitet werden, aber ein pfiffiger Elektronik-Hacker könnte auf die Idee kommen, die RS232 Schnittstelle mit eigener Elektronik zu "belauschen" und so in Kenntnis unseres Codes kommen.

Des Weiteren können beliebig viele Zahlenkombinationen in gleichen zeitlichen Abstand zum Ausprobieren eingeben werden. Die ermöglicht z.B. Brute-Force Angriffe auf das Codeschloss. Wenn also es jemanden gelingen sollte, an die Standard-RS232 Schnittstelle eine eigene Elektronik anzuschließen, die die Kommunikation zwischen Eingabe und Steuerungsteil aufzeichnet, könnte so der korrekte Eingabecode ermittelt werden. Diese "Hintertür" werden wir im heutigen Teil zwar nicht ganz schließen können, aber es unseren Elektronik-Hacker zumindest erheblich schwerer machen!
Um die genannten Schwächen des Systems im heutigen Teil zumindest anzugehen, implementieren wir als erstes zur Verzögerung bzw. zur Erschwerung eines Brute Force Angriffs eine Zeitverzögerungseskalation.
Eine Zeitverzögerungseskalation erhöht grundsätzlich mit jeder Falscheingabe des Codes die Zeit bis zur Eingabemöglichkeit eines weiteren Codes. Bei unserem Codeschloss wird die Eingabezeit bis auf eine Minute pro Eingabeversuch verlängert. Wer hier eigene Zeiten einrichten möchte, der kann dies in der Zeile im Controller B:

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

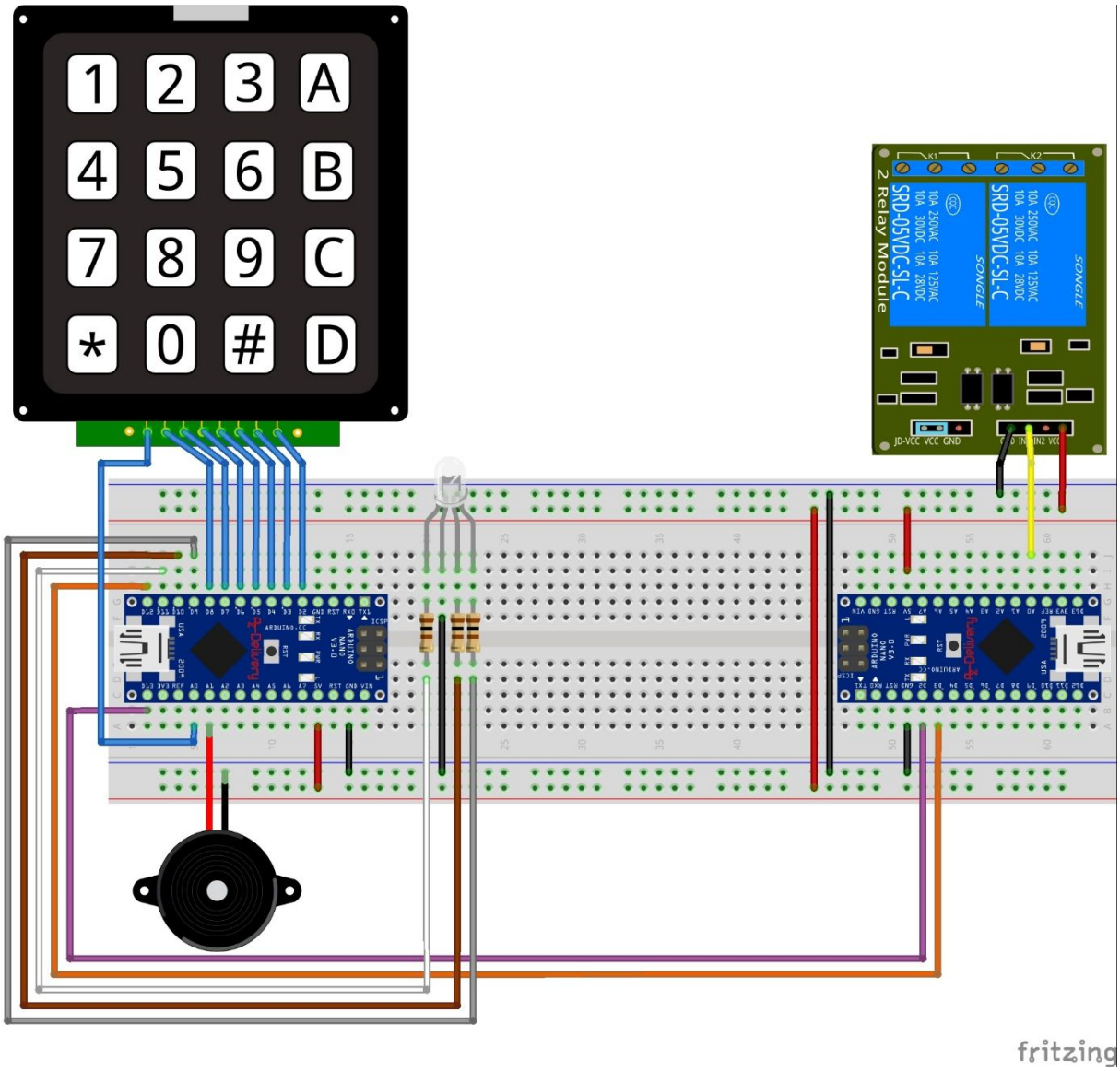
eigene Zeiten in Sekunden durch Anpassen der Werte definieren.

Damit ist bei genügend langem Code (ab 12-stellig) die benötigte Zeit zum durchprobieren aller möglichen Kombinationen so extrem lang, dass ein Brute Force Angriff nicht mehr in vernünftiger Zeit erfolgreich ist.
Zu dem zweiten angesprochenen Problem heißt unsere Lösung Verschlüsselung. Wir lassen die Arduinos die Kommunikation zwischen der Eingabe und Auswerteteil unseres Codeschlosses verschlüsseln! Bevor wir jedoch ins praktische gehen, tauchen wir gemeinsam kurz in die Basis der Kryptographie und binären Logiken ein. Insbesondere schauen wir uns in diesem Zusammenhang die binäre Operation "exklusives oder" (XOR) an, da diese für unser Vorhaben hervorragende Dienste leistet. Mittels einer XOR Verknüpfung ist eine einfache Verschlüsselung und Entschlüsselung möglich. Eine Erklärung der Verwendung von XOR in der Kryptographie sowie einer grundsätzlichen Erklärung der Funktionsweise finden Sie auf der Seite; https://www.archicrypt.de/live8/xor.htm. In vielen Lehrbüchern zum Thema Kryptographie ist zu lesen, dass eine XOR Verschlüsselung solange sicher ist, solange sich das sog. Schlüsselbyte sich für jedes zu verschlüsselnden Bytes (Klartext) ändert. Um sich daran zu nähern, ohne jedoch einen sehr aufwendigen und rechenintensiven Code schreiben zu müssen, bedienen wir uns nun einem kleinen Trick. Wir benutzen den eingebauten Pseudo-Radom Generator (deterministischen Zufallszahlengenerator) unseres Arduinos mit einem gleich auf beiden Systemen (Eingabe und Ausgabeeinheit) festgelegten Startwert! Mehr Informationen zum Thema nicht-deterministischen und deterministischen Zufallszahlengeneratoren finden Sie auf der Wikipedia Seite https://de.wikipedia.org/wiki/Zufallszahlengenerator. Wir nutzen für unser Codeschloss die Tatsache, dass die Funktion randomSeed() die auf zwei unterschiedlichen! Arduinos mit dem gleichen Startwert (Seed) gestartet wurde, die gleichen „Zufallszahlen“ erzeugt. Diesen doch recht interessanten und wissenswerten Fakt nutzen wir nun aus, indem wir die erzeugten (pseudo Zufalls-) Zahlen mit dem eigegebenen Zahlenwert der Tastatur XOR-Verknüpfen und so über die RS232 Leitung senden. Auf der Gegenseite machen wir zum "entschlüsseln" das gleiche mit der ebensolchen bekannten erzeugten Zufallszahl. Wir erhalten somit wieder den Ursprungswert (den Klar Wert) und können so die weitere Verarbeitung starten.

Kommen wir aber nun zum Aufbau unseres heutigen Projekts. Wir benötigen im heutigen Teil an Hardware:

Anzahl	Bezeichnung	Anmerkung
1	Relais Modul	
2	Arduino Nano	
1	4x4 Keypad	
3	Widerstände 120 Ohm	
1	RGB Led	

Wir bauen die Schaltung wie auf folgender gezeigter Fritzing Zeichnung auf und verbinden die beiden Controller über eine Softwareseitige Serielle Schnittstelle:



Zu beachten ist das TX des Controllers 1 an RX des Controllers 2 und vice versa angeschlossen werden muss.

```
TX <-> RX
RX <-> TX
```

Nachdem wir alles verkabelt haben können wir nun folgenden Code auf Controller A (Eingabeteil) hochladen:

```
// 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, 13); // RX, TX

char keys[ROWS][COLS] = {
                    {1,2,3,13},
                    {4,5,6,14},
                    {7,8,9,15},
                    {10,11,12,16},
                   };
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 CodeEnterSeqence = false;
bool CodeEnterSeqenceOLD = false;
bool InputBlocked = false;
bool PinEnteredFalseBefore  = false;
bool RGBFadeEnabled = true;
long previousMillis = 0;
long previousMillisKeyBoard = 0;
byte EnCodedKeyStroke = 0;
byte inByte = 0;
int CyclesInBlack = 0;
unsigned long InititalKey = 902841;

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

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); // NORMAL MODE
 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)
  {
  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;
 }
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)
 {
 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;
 }
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)
 {
 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;
  }
 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 == 20) // Code Correct
    {
    RGBControl(RGBGREEN,false);
    tone(PIEZOSUMMER, 1200, 200);
    delay(2000);
    PinEnteredFalseBefore  = false;
    RGBControl(RGBBLUE,true); // NORMAL MODE
    } else
    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;
    }
  }
}

void TimeMgmnt ()
{
if ((millis() - previousMillisKeyBoard > KeybModeTimeInterval1) & CodeEnterSeqence & 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
    }
  CodeEnterSeqence = false;
  previousMillisKeyBoard = millis();
  byte randNumber = random(0, 254);
  EnCodedKeyStroke = 10 ^ 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 == 10) | (key == 12))
   {
     RGBControl(RGBSHORTBLACK,true);
     tone(PIEZOSUMMER, 988, 100);
     delay(120);
     CodeEnterSeqence = false;
     if(key == 10)
      {
```

```
        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);
       CodeEnterSeqence = true;
       RGBControl(RGBCYAN,true);
       previousMillisKeyBoard = millis();
      }
    }
  }
}

void loop()
{
if (InSync)
  {
  KeyboardHandler(InputBlocked);
  }
TimeMgmnt ();
SerialHandler ();
}
```

Nun kann der nun folgenden Code auf Controller B (Auswerteeinheit) hochladen werden:

```
#include <SoftwareSerial.h>
#define RELAIS_A A0
#define Interval1 1000
#define MAXDelayStages 7
const byte MaxPinCodeLength = 20;
const byte DelayInterationsInSec[MAXDelayStages] = {1,5,10,20,30,45,60};

SoftwareSerial mySerial(2, 3); // RX, TX

byte KeyPadBuffer[MaxPinCodeLength];
byte PinCode[MaxPinCodeLength] = {1,2,3,13}; // Standard Pincode: 123A  - Bitte Ändern gemäß
Beschreibung -
byte BufferCount = 0;
byte a;
bool InSync = true;
bool AcceptCode =false;
byte ErrorCounter = 0;
long previousMillis = 0;
byte InputDelay = 0;
byte RecInititalKeyLength = 0;
```

```
unsigned long CommuncationKey = 902841;

void setup()
{
 Serial.begin(9600);
 mySerial.begin(9600);
 pinMode(RELAIS_A,OUTPUT); //Relais Output
 digitalWrite(RELAIS_A,HIGH); //LOW Aktiv
 BufferCount = 0;
 for (a = 0; a <= MaxPinCodeLength -1 ; a++)
  {
  KeyPadBuffer[a] = 0;
  }
 randomSeed(CommuncationKey);
}

void loop()
{
if (mySerial.available())
  {
   byte randNumber = random(0, 254);
   byte key = mySerial.read();
   byte DeCodedKeyStroke = key ^ randNumber;
   if ((DeCodedKeyStroke < 18) & InSync)
    {
     if(DeCodedKeyStroke == 10)   // Clear Keypad Buffer  Key: *
      {
       for (a = 0; a <= MaxPinCodeLength -1; a++)
        {
        KeyPadBuffer[a] = 0;
        }
       Serial.print("Clear ");
       Serial.println(BufferCount);
       mySerial.write(23);
       BufferCount = 0;
      } else
     if(DeCodedKeyStroke ==12)   // Enter Keypad Buffer  Key: #
      {
      if (InputDelay == 0)
       {
       Serial.println("Auswertung gestertet");
       Serial.println(BufferCount);
       AcceptCode = true;
       for (a = 0; a <= MaxPinCodeLength -1 ; a++)
         {
         if (!(PinCode[a] == KeyPadBuffer[a])) {AcceptCode = false; }
         Serial.print(PinCode[a]);
         Serial.print(";");
         Serial.print(KeyPadBuffer[a]);
         Serial.println(" ");
         }
       Serial.println("END");
       if (AcceptCode)
```

```
        {
        mySerial.write(20);
        digitalWrite(RELAIS_A,(!digitalRead(RELAIS_A)));
        ErrorCounter = 0;
        InputDelay = 0;
        AcceptCode = false;
        } else
        {
        mySerial.write(21);
        if ( ErrorCounter < MAXDelayStages - 1) { ErrorCounter++; }
        InputDelay = DelayInterationsInSec [ErrorCounter];
        }
      for (a = 0; a <= MaxPinCodeLength -1; a++) { KeyPadBuffer[a] = 0; }
      Serial.println("Clear all Memory");
      BufferCount = 0;
      } else
      {
       Serial.println("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("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 ("release");
    mySerial.write(40);  // Delay Mode End
    }
   InputDelay = InputDelay - 1;
   }
```

```
  }
}
```

Nun können wir durch die Eingabe des Codes 123A das Relais ein und ausschalten. Der Pincode zum schalten des Relais kann in der Zeile:

```
byte PinCode[MaxPinCodeLength] = {1,2,3,13}; // Standard Pincode: 123A
```

Im Code von Controller 2 (Logik- und Steuerungsteil) geändert werden. Leider gibt es auch heute wieder einen Schwachpunkt bei unserem Codeschloss: Kennt man die über RandomSeed übergebene Initialnummer, kann man die Verschlüsselung aufbrechen. Diese könnte man z.b Durch ein auslesen des Roms der Eingabeeinheit in Erfahrung bringen. Wie wir das Auslesen der Zahl aus dem ROM verhindern können und so unserem Hacker das Leben wieder ein bisschen schwerer machen können, erfahrt Ihr im nächsten Teil!

Bis dahin wünsche ich viel Spaß beim Nachbauen!