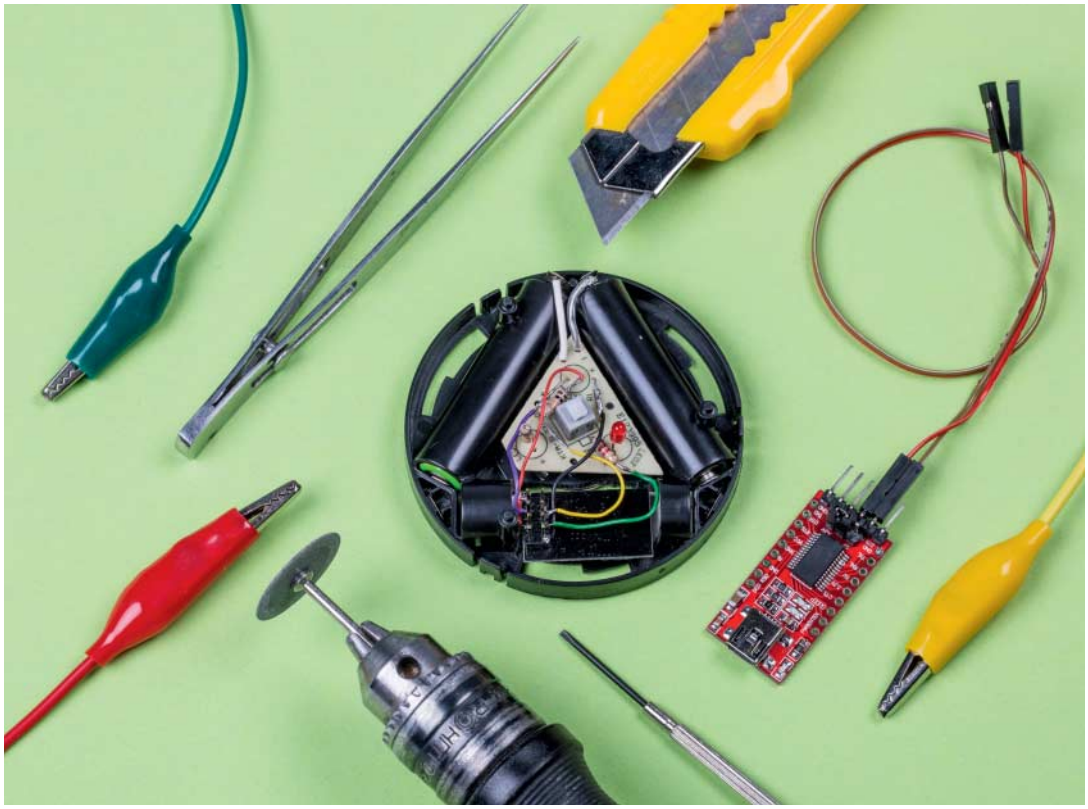


Ein ESP-01, ein Taster, drei Widerstände und eine LED, frei verdrahtet auf der zweckentfremdeten Platine einer Batterielampe: Fertig ist der Selbstbau-Dash-Button.

Verlasst die Ama-Zone!

Batteriebetriebene IoT-WLAN-Buttons selbst bauen und programmieren

Amazon verlangt für seinen WLAN-fähigen Dash-Button 5 Euro. Mit dem ESP8266 bauen Sie einen eigenen Button für 3,77 Euro, der sich schneller verbindet und garantiert keine Amazon-Server kontaktiert. Und programmieren können Sie ihn auch.

Von Johannes Merkert

Das IoT-Gerät des Jahres 2016 war Amazons Dash-Button. Alle Medien diskutierten die fünf Euro teuren Bestellknöpfe, entweder als Vorreiter eines ungezügelten Konsumwahns, als Einkaufsmethode der Zukunft oder als das endgültige Ende des Datenschutzes. Im Internet tauchten währenddessen zahlreiche Ideen auf, wofür man einen batteriebetriebenen Button abseits der von Amazon intendierten Bestellfunktion nutzen könnte. Der Nachteil dieser Hacks: Der Amazon Dash-Button spricht immer mit Amazon-Servern, wenn er sie erreichen kann. Wir haben eine Lösung vorgestellt, mit der Sie Ihre Buttons mit einem Raspi von Amazon abtrennen können [1]. Eleganter ist aber ein Knopf, bei dem Sie selbst entscheiden, mit welchem Server er spricht.

Genau dafür eignet sich das in der Maker-Szene beliebte Modul ESP-01 mit dem Mikrocontroller ESP8266. Seine 80-MHz-CPU hat genug Dampf für verschlüsselte Kommunikation per TLS 1.1 und sein Schlafmodus verbraucht so wenig Energie, dass der Button mit zwei AAA-Batterien jahrelang läuft. Die Platine mit SoC, Antenne und Stiftleiste zum Anschließen ist nur so groß wie eine Fingerkuppe und passt problemlos in das Gehäuse von batteriebetriebenen LED-Schranklichtern (Stichwort: Stick & Push). Mit einem USB-Seriell-Konverter für nicht mal zwei Euro schließen Sie das ESP-Modul am Rechner an und programmieren es mit der Arduino-Entwicklungsumgebung (IDE). Die selbst geschriebene Firmware in C kommt dank guter Bibliotheken für den ESP8266 mit nur 150 Zeilen Quelltext aus.

Ausstattung

Um einen Dash-Button selbst zu bauen, brauchen Sie einen Lötkolben, ein Cuttermesser und eine Laubsäge, um das Plas-

Einkaufsliste

- ESP-01 (1,70 €)
- 2 × Widerstand 10 kΩ (0,30 €)
- Widerstand 2,2 kΩ (0,08 €)
- Low-Current-LED (0,09 €)
- Stick&Push-LED-Leuchte (1 €)
- 2 × AAA-Batterie (0,60 €)

Programmer
- USB-UART-Interface (1,50 €)
- Dupont/Jumper-Kabel (1 €)

optional
- Breadboard (3 €)
- Buchse 4 × 2 (0,20 €) oder ESP-01-Breadboard-Adapter (1,30 €)
- 2 × Mikrotaster (0,40 €)
- 3,3-V-Spannungswandler fürs Breadboard (1 €)
- Netzteil (5 €)

tikgehäuse des Schranklichts zu modifizieren. Außerdem müssen Sie sich ein paar Stunden Zeit nehmen.

In der Einkaufsliste finden Sie die Bauteile, die Sie für jeweils einen Button brauchen. Das serielle USB-UART-Interface (Universal Asynchronous Receiver Transmitter) und die Steckkabel brauchen Sie nur einmal, egal wie viele Buttons Sie bauen. Falls Sie mit diesem Artikel in eigene Experimente mit dem ESP8266 einsteigen möchten, lohnt sich ein Breadboard samt 3,3-V-Stromversorgung.

Die angegebenen Preise beziehen sich auf die günstigsten Bezugsquellen, beispielsweise bei Ebay. Die versenden oft direkt aus China, was nur gut geht, weil der Warenwert unter der Bagatellgrenze des Zolls liegt. Außerdem dauert der Versand aus China ungefähr drei Wochen. Wenn Sie gezielt nach Angeboten aus Deutschland suchen, bekommen Sie die Platinchen innerhalb weniger Tage. Allerdings zahlen Sie dann circa 5,50 Euro pro Button und 4,20 Euro für Interface und Kabel.

Hardware

Der Selbstbau-Dash-Button soll wie das Original mit Batterien laufen. Erst die Unabhängigkeit von der Steckdose macht den Knopf attraktiv für viele Aufgaben. Der ESP8266 ist auf 3,3 V Betriebsspannung ausgelegt. Er läuft aber auch mit 3 V einwandfrei, sodass Sie ihn einfach mit zwei in Reihe geschalteten AAA-Batterien betreiben können. Vorsicht: Die 4,5 V von drei Batterien oder 5 V wie bei Arduino-Basteleien zerstören den Chip.

Bei der Suche nach einem passenden Gehäuse für zwei Batterien und einen Knopf stießen wir auf LED-Leuchten für drei AAA-Batterien, die als Nachtlicht oder Schrankbeleuchtung beworben werden. Die etwa handgroßen runden Leuchten gibt es ab 1 Euro. Sie bestehen fast komplett aus einem glasklaren Schalter mit drei in einem Reflektor eingebetteten LEDs. Die LEDs sitzen zusammen mit dem Ein-/Ausschalter auf einer kleinen dreieckigen Platine zwischen den drei Batterien.

Der Ein-/Ausschalter besitzt eine Mechanik ähnlich der im Kugelschreiber, die ihn nach dem ersten Druck im eingeschalteten Zustand festhält. Verantwortlich dafür ist ein kleines Drähtchen im Gehäuse des Schalters. Mit einem feinen Schraubendreher lassen sich die Befestigungslaschen dieses Gehäuses öffnen und das Drähtchen mit einer Pinzette oder einer kleinen Zange entfernen. Damit bauen Sie den Ein-/Ausschalter zum Taster um.

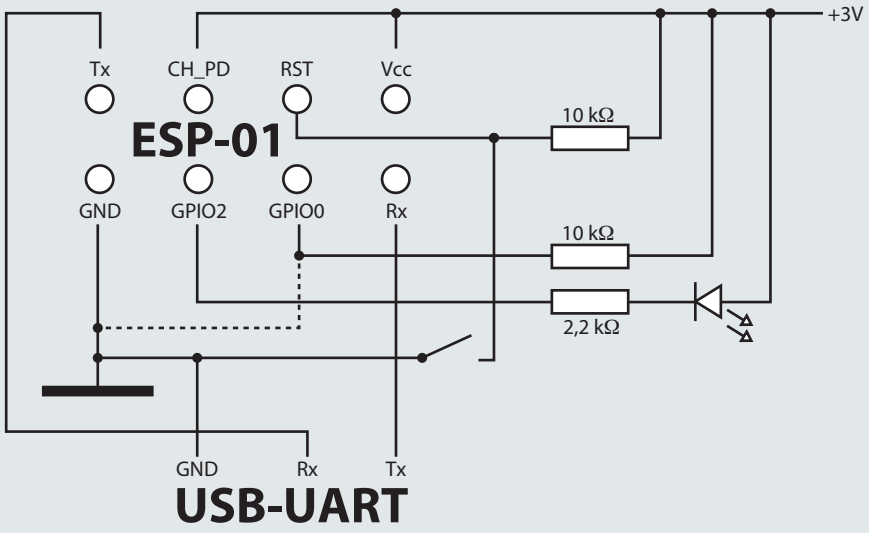
Da der Prozessor nur mit zwei Batterien laufen soll, können Sie anstelle des Fachs der dritten Batterie Platz für das ESP-01-Modul schaffen. Wir haben dafür die Laubsäge angesetzt. Das ESP-01 ist nämlich etwas breiter als eine AAA-Batterie.

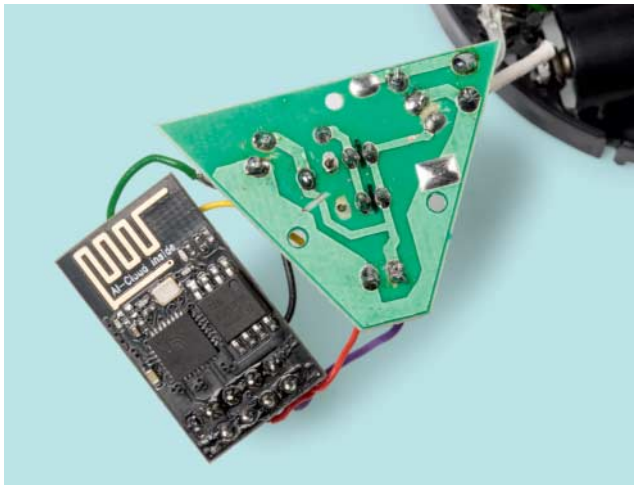
Anschließend rücken Sie der dreieckigen Platine mit dem Lötkolben zu Leibe: Die Power-LEDs müssen raus. Sie verbrauchen zu viel Strom. An der Position einer der LEDs setzen Sie eine Low-Current LED (2 mA) ein, die den Status des Buttons an den Nutzer blinken soll. Sie sitzt in Reihe zu dem 2,2-kΩ-Vorwiderstand. Einer der 10-kΩ-Widerstände zieht den Pegel von IO-Pin 0 auf 3 V, um den ESP in den normalen Betriebsmodus zu versetzen. Der zweite zieht den Reset-Pin auf 3 V. So lange der ESP läuft und Reset an 3 V liegt, startet er nicht neu. Am Reset-Pin hängt zusätzlich der Taster, der ihn mit Masse verbindet. Wird der Taster gedrückt, kommen keine 3 V mehr am Reset-Pin an und er liegt stattdessen auf Masse-Pegel. Diese Reset-Masse-Verbindung löst den Neustart des Mikrocontrollers aus.

Um diese Schaltung aufzubauen, können Sie die dreieckige Platine weiter ver-

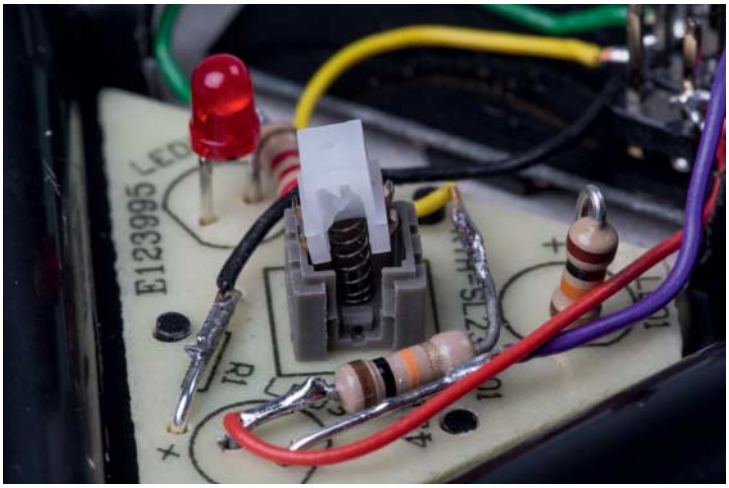
Schaltplan

Von den 8 Pins des ESP-01 dienen zwei als Datenleitung für die Programmierung. Den Rest verbinden Sie wie abgebildet mit LED, Taster und Spannungsversorgung. Die gestrichelte Verbindung sorgt dafür, dass der ESP8266 im Programmiermodus startet.





Die Platine der LED-Lampe können Sie mit leichter Modifikation weiter verwenden: Wir mussten eine Leiterbahn durchtrennen und ein zusätzliches Loch bohren.



Den Ein-/Ausschalter des Schranklichts bauen Sie zum Taster um, indem Sie vorsichtig die aufgeklipste Abdeckung entfernen und ein Drähtchen herausziehen.

wenden. Für unseren Knopf reichte es, eine Leiterbahn an einer Stelle zu unterbrechen, ein zusätzliches Loch zu bohren und auf der Oberseite einen Draht einzufügen. Je nach Hersteller unterscheidet sich das Platinenlayout Ihres LED-Lichts möglicherweise leicht von unserem.

Die Verbindungen zum ESP-01 stellen Sie beispielsweise über Kupferlackdraht oder kurze Litzenstücke her, die Sie am unteren Ende der Pins anlöten sollten. Zum Programmieren müssen Sie nämlich auf den Masse-Pin zusätzlich ein Jumper-Kabel stecken.

Programmiermodus

Zum Programmieren verbinden Sie den ESP-01 mit dem seriellen USB-Interface. Achten Sie darauf, dass das Interface nur 3,3 V an die Datenleitungen anlegt. Einige lassen sich zwischen 3,3 V und 5 V umschalten. Sorgen Sie mit einem Jumper-Kabel für eine gemeinsame Masse und verdrahten mit zwei weiteren Jumper-Kabeln jeweils Rx mit Tx und umgekehrt. Verbinden Sie das Interface zuletzt per USB mit dem Rechner.

Mit diesem Aufbau können Sie beispielsweise mit dem Java-Tool ESPlorer (siehe c't-Link) eine Verbindung zur vorinstallierten AT-Firmware auf dem ESP aufbauen. Um die Firmware durch Ihr eigenes Programm zu ersetzen, versetzen Sie den ESP zuerst in den Programmiermodus.

Den erreichen Sie, wenn der GPIO 0 bei einem Reset auf Masse liegt. Verbinden Sie dafür beispielsweise mit einem Kabel mit Krokodilklemmen auf der Platine GPIO 0 mit Masse. Achten Sie darauf, dass die Verbindung vor dem 10-kΩ-Widerstand liegt, weil Sie sonst einen Kurzschluss verursachen. Drücken Sie dann den Taster, der den Reset auslöst.

Im Prinzip können Sie die Verbindung zwischen GPIO 0 und Masse jetzt entfernen. Der ESP bleibt zunächst im Programmiermodus, bis Sie ein Programm übertragen haben oder erneut resetten. Sofort nach einer erfolgreichen Übertragung fährt er jedoch im normalen Modus hoch (unabhängig vom Pegel an GPIO 0) und führt das übertragene Programm aus. Das ist sehr praktisch, da Sie das Programm so immer gleich nach der Übertragung automatisch testen können. Fällt Ihnen dabei ein Fehler auf, kommen Sie mit einem Reset direkt wieder in den Programmiermodus, falls Sie die Masseverbindung an GPIO 0 nicht entfernt haben.

Arduino-IDE

Der Arduino-IDE bringen Sie den Umgang mit dem ESP8266 bei, indem Sie zunächst den „Boards-Manager“ erweitern. Tragen Sie dafür unter „Datei/Voreinstellungen“ in die Zeile „Additional Boards Manager URLs“ folgende URL ein:

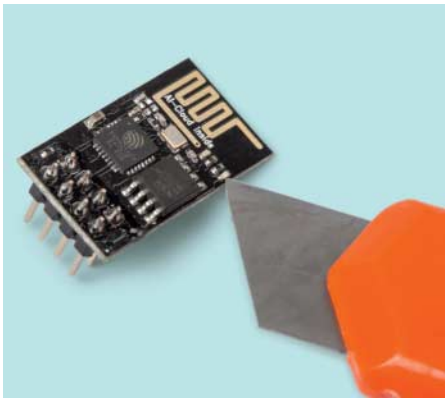
```
http://arduino.esp8266.com/stable/package_esp8266com_index.json
```

Öffnen Sie danach den Boards-Manager („Werkzeuge/Platine/Boards Manager“) und geben ins Suchfeld „esp“ ein. Beim einzigen Eintrag „esp8266 by ESP8266 Community“ drücken Sie dann auf den Installieren-Button.

Nach der Installation wählen Sie unter „Werkzeuge/Platine“ „Generic ESP8266 Module“ aus, stellen „Upload Speed“ auf 115200 und „Port“ auf die erste USB-Schnittstelle. Wer herumprobieren möchte, findet unter „Datei/Beispiele“ auch ein Untermenü für den ESP8266 mit Beispielprogrammen zu den häufigsten Anwendungsfällen. Unsere Firmware laden Sie aus dem Repository (siehe c't-Link) oder klonen sie mit Git direkt ins Sketch-Verzeichnis:

```
git clone https://github.com/pinae/ESP8266-Dash.git Arduino/
```

Der komplette Quellcode liegt in der Projektdatei ESP8266-Dash.ino.



Leider sitzt auf dem ESP-01 eine LED, die immer leuchtet, wenn das Modul mit Strom versorgt wird. Sie allein leert die Batterien innerhalb weniger Tage. Mit der Spitze eines Cuttermessers können Sie die LED leicht zerbrechen und der Stromverschwendung ein Ende setzen.

Die eigene Firmware

Der Blick in den Quelltext (ESP8266-Dash.ino) zeigt, wie wenig dort wirklich passiert. In der ESP8266WiFi-Bibliothek steckt der komplizierte Teil der WLAN-Kommunikation, während das Hauptprogramm nur den groben Ablauf definiert.

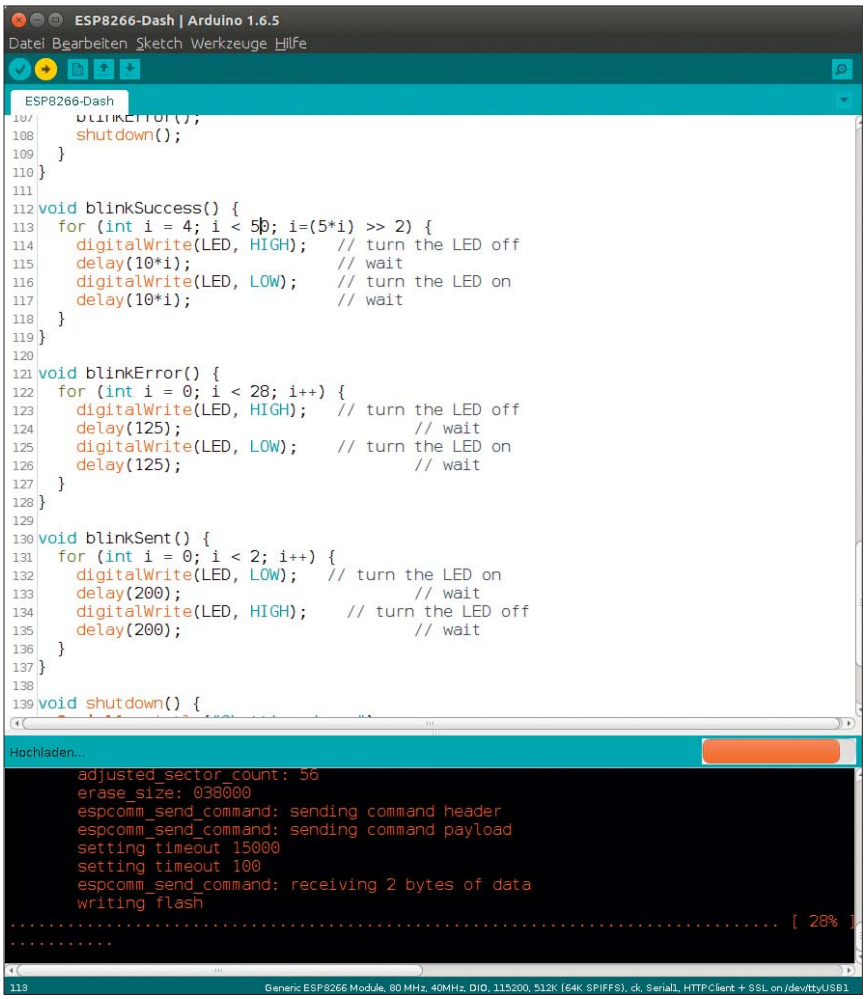
Der beginnt in der Funktion `setup()` in Zeile 21. Zunächst konfiguriert die Firmware GPIO 2 mit der Funktion `pinMode()` als Ausgang und setzt ihn mit `digitalWrite()`.

Anschließend baut die Firmware die WLAN-Verbindung auf:

```
WiFi.begin(WLAN_SSID, WLAN_PASS);
```

`WLAN_SSID` und `WLAN_PASS` sind in Zeile 6 und 7 fest definierte Strings, die Sie passend zu den Zugangsdaten Ihres WLAN anpassen müssen. Der Verbindungsaufbau dauert etwa zwei Sekunden, sodass das Programm so lange warten muss, um die Verbindung anschließend nutzen zu können. Ob der Verbindungsaufbau abgeschlossen ist, erkennt der Code daran, dass `WiFi.status()` den Wert `WL_CONNECTED` annimmt. Damit das Programm nicht ewig nach einem fehlenden Netzwerk sucht, legt es sich nach 30 Sekunden ohne Verbindung schlafen:

```
int failcounter = 300;
while(WiFi.status() != WL_CONNECTED) {
    delay(100);
    if(failcounter <= 0) {
      blinkError();
      shutdown();
    }
    failcounter--;
}
```



Mit der Arduino-IDE programmieren Sie den ESP8266 genauso leicht wie den Arduino. Kompilierung und Upload klappen dank des gut integrierten ESP-Plug-ins mit einem Klick.

Die Funktionen `blinkError()` und `shutdown()` definiert der Quellcode in Zeile 124 und 142. Die Blinkfunktion bringt die LED hektisch für 7 Sekunden zum Blinken. `shutdown()` legt den Mikrocontroller mit `ESP.deepSleep(0)` bis zum nächsten Reset in einen Tiefschlafzustand, in dem er maximal 10 µW verbraucht.

Anschließend kommt der `WiFiClientSecure` zum Einsatz, der in Zeile 17 der Variablen `client` zugewiesen wird. Die Methode `client.connect()` baut zu dem übergebenen Host eine Verbindung auf. Sie meldet zurück, ob der angegebene Host prinzipiell Anfragen akzeptiert. Eine SSL-Verbindung zum Raspi mit der IP `192.168.12.1` bauen Sie folgendermaßen auf:

```
const char* host = "192.168.12.1";
int port         = 443;
if (client.connect(host, port)) {
```

Bei einer solchen Verbindung antwortet der Server zunächst mit seinem Zertifikat, das normale Clients über die Zertifikatskette bis zu einer Liste vertrauter CAs prüfen. Dem ESP8266 fehlt jedoch die Rechenleistung für so eine aufwendige Prüfung und der Speicherplatz für eine umfangreiche Liste von CAs. Deswegen kürzen Sie die Zertifikatsprüfung ab, indem Sie lediglich prüfen, ob der SHA1-Hash des Zertifikats mit einem fest einprogrammierten Wert übereinstimmt. Damit pinnen Sie auch gleichzeitig das Server-Zertifikat [2]. Die Prüfung übernimmt die Funktion `verifyFingerprint()`:

```
void verifyFingerprint() {
  if (! client.verify(FINGERPRINT,
                      CERT_NAME)) {
    // Connection insecure!
    blinkError();
    shutdown();
  }
}
```

Die Konstanten `FINGERPRINT` und `CERT_NAME` definiert das Programm in Zeile 11 und 12. `CERT_NAME` muss genau dem Common Name in Ihrem Zertifikat und `FINGERPRINT` dem SHA1-Hash entsprechen. Den Hash geben Sie als paarweise gruppierte Zeichenkette mit hexadezimalen Ziffern an.

Diese Art der Zertifikatsprüfung hat den Nachteil, dass sich das Zertifikat nicht ändern darf. Wenn Sie also auf dem Server, mit dem sich der Button verbindet, ein anderes Zertifikat verwenden möch-

ten, müssen Sie auch die Firmware des Buttons anpassen und neu flashen.

Falls Sie darauf angewiesen sind, dass sich das Zertifikat ändern darf, müssen Sie die Firmware so erweitern, dass Sie nacheinander mehrere Hashes prüft und die Liste der Hashes zusätzlich austauschen kann. Das hätte unser kleines Beispiel aber zu sehr aufgebläht.

Nach der Zertifikatsprüfung können Sie guten Gewissens Requests über die verschlüsselte Verbindung schicken. Ein leerer GET-Request an die Default-Seite des Hosts sieht beispielweise so aus:

```
client.print(String("GET /") +
             " HTTP/1.1\r\n" +
             "Host: " + host + "\r\n" +
             "Connection: close\r\n" +
             "\r\n");
```

Einen Augenblick später schickt der Server seine Antwort zurück. Wir haben uns darauf beschränkt, darin eine Zeile zu suchen, die mit „OK“ beginnt, um in diesem Fall davon auszugehen, dass der Tastendruck erfolgreich registriert wurde. Sobald der Server die Verbindung abbricht, hat der Button die komplette Antwort empfangen. Ähnlich wie beim Aufbau der WLAN-Verbindung verhindert der Code in jedem Fall eine Endlosschleife:

```
int success = 0;
failcounter = 10000;
while (client.connected()) {
  if (client.available()) {
    String line = client
                .readStringUntil('\n');
    if (line[0] == 'O' &&
        line[1] == 'K') {
      digitalWrite(LED, LOW);
      success = 1;
    }
  }
  if (failcounter <= 0) {
    blinkError();
    shutdown();
  }
  failcounter--;
}
```

Danach signalisiert das Programm je nach `success` den Erfolg mit `blinkSuccess()` oder den Fehler mit `blinkError()`. Bei der Funktion `blinkSuccess()` haben wir uns den Spaß gemacht, die Blinkfrequenz von schnellem zu langsamem Blinken zu verändern. Das soll verdeutlichen, dass sich die Daten erfolgreich vom Button weg bewegt haben.

Selbstbau IoT

Der Selbstbau-Button ist nicht nur billiger als Amazons Original, sondern auch schneller. Bei unserem Hack mit dem Amazon-Button konnte unser Raspi erst nach circa 5 Sekunden auf Tastendrücke reagieren, während ihn nach weniger als 3 Sekunden die Pakete unseres Selbstbau-Buttons erreichten.

Bei der Energieaufnahme haben wir 0,5 mWh pro Tastendruck gemessen. Mit üblichen AAA-Batterien sollten damit circa 2000 Tastendrücke möglich sein – genug für mehrere Jahre Betrieb ohne Batteriewechsel. Der Standby-Verbrauch lag unterhalb unserer Messmöglichkeiten. Die vom ESP-Hersteller Espressif angegebenen 10 µW versprechen ebenfalls jahrelangen Betrieb. Aber auch wer sehr viel drücken möchte, braucht keine Angst zu haben: Im Gegensatz zum verklebten Amazon-Button können Sie die Batterien beim Selbstbau-Button leicht wechseln.

Damit brauchen Sie jetzt nur noch kreative Ideen, was Sie mit einem WLAN-Button alles schalten möchten. Er könnte an der Waschmaschine kleben und „Waschmittel“ auf Ihre Einkaufsliste schreiben. Das erlaubt Ihnen, Preise zu vergleichen und zwingt Sie nicht automatisch zur Online-Bestellung. Oder Sie zählen damit, wie oft Sie den Heimtrainer benutzen oder den Kühlschrank aufmachen. Für den Kühlschrank können Sie den ESP8266 auch direkt in die Tür einbauen. Oder Sie aktivieren mit dem Button die Kaffeemaschine für die erste Tasse nach dem Aufstehen bequem vom Bett aus. Sollten Sie noch keine WLAN-Kaffeemaschine haben, bauen Sie doch Ihre bestehende um: Sie kennen ja jetzt den ESP8266 und wissen, wie man ihn programmiert. *(jme@ct.de)* c't

Dem fertigen Knopf sieht man sein selbst gebautes Innenleben kaum noch an.

Literatur

[1] Johannes Merkert, Drückerei, Den Amazon Dash Button zweckentfremden, c't 21/16, S. 174
[2] Jürgen Schmidt, Festgenagelte Zertifikate, TLS wird sicherer durch Certificate Pinning, c't 23/15, S. 118

Downloads usw.: ***ct.de/yj3v***

Serverkonfiguration

Leider unterstützt der ESP8266 bisher nur verschlüsselte Verbindungen mit TLS 1.1, RSA-Schlüsselaustausch und 128- oder 256-Bit-AES-Verschlüsselung. Damit ist keine Perfect Forward Secrecy möglich, sodass viele Administratoren diesen Betriebsmodus in ihren Konfigurationen nicht erlauben. Für einen Webserver nach modernstem Standard sollten Sie das auch nicht erlauben, da Ihre Verbindungen von Angreifern auf den RSA-Schlüsselaustausch heruntergestuft werden könnten. Unsere Prototypen haben wir so programmiert, dass sie sich mit einem Nginx-Webserver auf einem Raspberry Pi im gleichen Netz verbunden haben. Die für den Button nötigen Kryptoverfahren aktivieren Sie bei Nginx mit folgenden beiden Zeilen:

```
ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
ssl_ciphers 'EECDH+AES:EDH+AES:
↳RSA+AES:!LOW:!3DES:!MD5:!EXP:!PSK:
                    ↳!DSS:!RC4';
```

Den vom Button benötigten Modus aktiviert die Angabe `RSA+AES` in der Cipher-Liste für OpenSSL. Bei Apache 2 verläuft die Konfiguration analog.