Sichere Speicherung geheimer Schlüssel auf einem Raspberry Pi

Studienarbeit

im Fach Informatik

Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik

eingereicht von: Alexander Fehr

Betreuer: Dr. Wolf Müller

Berlin, 19.09.2018

Inhaltsverzeichnis

1 Einführung **4**

2 Vorbetrachtungen **5**
2.1 KNX 5
2.2 KNX Data Secure 6
2.3 Raspberry Pi 7
2.4 Pigator 8

3 Anforderungen **10**
3.1 Funktionale Anforderungen 10
3.2 Anforderungen an die Kosten 11
3.3 Sicherheitsanforderungen 11
3.3.1 Schutzziele 11
3.3.2 Angreifermodell 11
3.3.3 Bedrohungsanalyse 12
3.4 Annahmen und Abgrenzung 12

4 Vergleich der Speicherlösungen **14**
4.1 Speicherung im Dateisystem 14
4.2 Speicherung im Dateisystem verschlüsselt mit Passwort 15
4.3 Speicherung auf einer Smartcard 15
4.4 Logische Speicherung auf einer Smartcard 17
4.5 Speicherung in einem HSM 18
4.6 Speicherung in einem TPM 20
4.7 Speicherung in der ARM TrustZone 21

5 Ausblick **23**

6 Fazit **24**

7 Literatur **25**

Abbildungsverzeichnis

1 Verbindung von Geräten über den KNX-Bus 5
2 KNX Data Secure Paket 6
3 Raspberry Pi 3 Model B+ 8
4 Pigator mit KNX-Modul 9
5 CryptoMate Nano USB-Token 16
6 Zymkey für den Raspberry Pi 19
7 LetsTrust TPM für den Raspberry Pi 20

1 Einführung

KNX ist ein langjähriger Standard für intelligente Heim- und Gebäudetechnik. Mit KNX können beispielsweise Lampen oder Heizungen zentral gesteuert werden. Erst in der letzten Zeit wurde mit KNX Data Secure eine Möglichkeit geschaffen die Kommunikation zwischen KNX-Geräten abzusichern und so Vertraulichkeit und Datenintegrität zu gewährleisten.

Ziel dieser Arbeit ist es verschiedene Möglichkeiten der Schlüsselspeicherung für ein Raspberry Pi-basiertes Intrusion Detection System (IDS) einer KNX Data Secure-Installation zu evaluieren. Der Sicherheit des IDS-Schlüsselspeichers kommt dabei eine besondere Bedeutung zu, da er im Gegensatz zu einzelnen KNX Data Secure-Geräten alle Schlüssel der überwachten KNX-Installation enthält. Eine Verletzung der Vertraulichkeit dieser Schlüssel hätte weitreichende Konsequenzen für die gesamte KNX-Installation.

Die vorliegende Arbeit ist in mehrere Kapitel gegliedert: Zunächst werden in Kapitel 2 einige Grundlagen bezüglich KNX und dem Raspberry Pi vermittelt. Anschließend werden in Kapitel 3 die Anforderungen an den Schlüsselspeicher aus verschiedenen Perspektiven beleuchtet, bevor dann in Kapitel 4 unterschiedliche Lösungen zur Speicherung vorgestellt und bewertet werden. Kapitel 5 gibt schließlich einen kurzen Ausblick wie das Thema sinnvoll fortgesetzt werden könnte.

2 Vorbetrachtungen

Bevor auf die Speicherung der Schlüssel eingegangen wird, werden nachfolgend einige in diesem Kontext wichtige Grundlagen vorgestellt. Dabei geht es einerseits um KNX und dessen Absicherung und andererseits um den Raspberry Pi und dessen Anbindung an den KNX-Bus.

2.1 KNX

KNX [1] ist ein weltweiter und offener Standard für Heim- und Gebäudeautomation. Die KNX-Spezifikationen werden von der KNX Association [2] entwickelt und sind kostenlos erhältlich. Mittels KNX lassen sich unter anderem Heizungs-, Beleuchtungs- und Zugangskontrollsysteme steuern. Dabei findet man KNX-Installationen sowohl in Einfamilienhäusern als auch in Büro- oder Industriegebäuden.

Die Geräte einer KNX-Installation werden über einen Feldbus miteinander verbunden (Abbildung 1), so dass sie Daten untereinander austauschen können. Dabei können Geräte unterschiedlicher Hersteller in einer Installation eingesetzt werden, sofern sie entsprechend durch die KNX Association zertifiziert sind. Mittlerweile existieren mehr als 7000 zertifizierte Produkte. Diese reichen von Systemgeräten wie Kopplern zum Verbinden von Busabschnitten über Sensoren wie Bewegungsmeldern bis hin zu Aktoren wie Lichtdimmern.

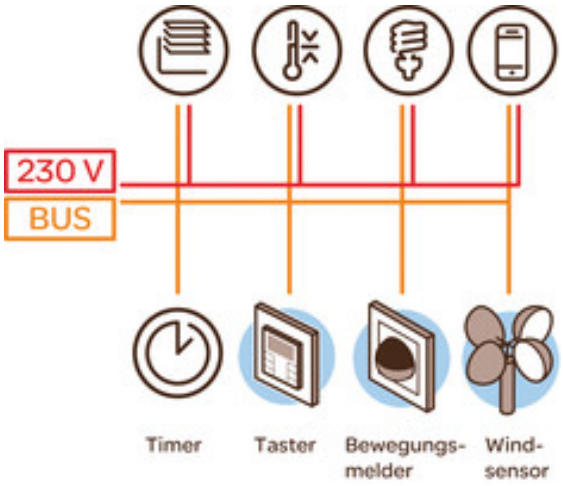

Abbildung 1: Verbindung von Geräten über den KNX-Bus (Quelle: [3])

Neben einem kabelgeführten Twisted-Pair-Bus (Übertragungsrate 9,6 kbit/s), gibt es

auch IP-basierte (KNXnet) und funkbasierte Lösungen (KNX-RF). Eine KNX-Installation wird in maximal 15 Bereiche mit jeweils 15 Linien und 256 Bus-Teilnehmern pro Linie aufgeteilt. Die Datenpakete auf dem KNX-Bus werden in Form von Telegrammen versendet. Die maximale Länge eines Standard-Telegramms beträgt 23 Oktetts und bei erweiterten Frames sogar bis zu 263 Oktetts. Quell- und Zieladressen der Telegramme werden in der Form `<Bereich>.<Linie>.<Teilnehmer>` angegeben. Außerdem können zusammengehörige Geräte auch gruppiert werden. Eine entsprechende Gruppenadresse hat die Form `<Hauptgruppe>/<Mittelgruppe>/<Untergruppe>`.

Die Konfiguration des KNX-Systems erfolgt mit der Windows-basierten Software ETS (Engineering Tool Software) [4]. ETS wird von der KNX Association weiterentwickelt und liegt aktuell in Version 5 vor. Es sind verschiedene Varianten verfügbar, welche sich durch die maximale Anzahl der konfigurierbaren Geräte unterscheiden.

2.2 KNX Data Secure

Auf dem KNX-Bus werden Daten unverschlüsselt übertragen. Besonders Bus-Leitungen die ins Freie führen (z. B. Außensensoren) stellen ein Sicherheitsrisiko dar. KNX-Systeme wie beispielsweise Alarmanlagen und Torsteuerungen können so mögliche Ziele für einen Angriff werden. Die KNX Association hat deshalb mit KNX Data Secure [5] eine Lösung geschaffen um die Laufzeitkommunikation auf dem KNX-Bus abzusichern.

Durch KNX Data Secure [6] werden Benutzerdaten durch Verschlüsselung und Authentifizierung vor unberechtigtem Zugriff und Manipulation geschützt. Dabei werden Datenintegrität (durch Authentifikationscodes), Vertraulichkeit (durch Verschlüsselung der Daten) und Freshness (durch Sequenznummern) sichergestellt. KNX Data Secure nutzt zur Verschlüsselung und MAC-Berechnung den AES-128 CCM Algorithmus. Für die Übertragung wird dabei ein längeres KNX-Telegrammformat verwendet (Abbildung 2), welches bis Oktett 5 wie ein Standard-Telegramm aufgebaut ist und ab Oktett 6 die gesicherte APDU enthält.

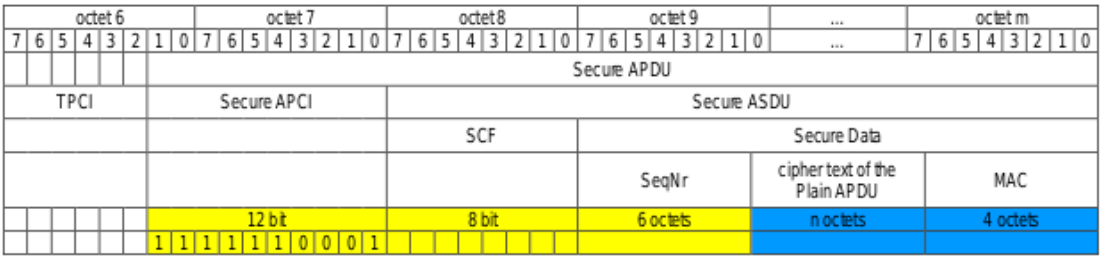

Abbildung 2: KNX Data Secure Paket (gelb=Plaintext, blau=verschlüsselter Text) (Quelle: [5])

Die Erzeugung der symmetrischen Schlüssel für die Laufzeitkommunikation erfolgt in einem mehrstufigen Prozess. Zunächst muss der mit jedem KNX Data Secure-fähigem Gerät ausgelieferte gerätespezifische FDSK (Factory Device Set up Key) in die ETS eingegeben werden. Daraufhin ezeugt die ETS einen gerätespezifischen Werkzeugschlüssel, welcher mit dem FDSK verschlüsselt und authentifiziert über den Bus an das Gerät gesendet wird. Das Gerät akzeptiert ab da nur noch den Werkzeugschlüssel für die weitere Kommunikation mit der ETS. Anschließend erzeugt die ETS die benötigten Laufzeitschlüssel und sendet diese mit den entsprechenden Werkzeugschlüsseln verschlüsselt und authentifiziert an die jeweiligen Geräte.

2.3 Raspberry Pi

Der Raspberry Pi [7] ist ein kreditkartengroßer Einplatinencomputer mit ARM-Prozessor. Er wurde von der Raspberry Pi Foundation [8] entwickelt, welche seit 2012 verschiedene Raspberry-Modelle auf den Markt gebracht hat. Ziel der Entwicklung war es, jungen Menschen einen günstigen Computer zum Erwerb von Programmier- und Hardwarekenntnissen zur Verfügung zu stellen. Der aktuelle Verkaufspreis beträgt etwa 35 EUR. Neben seiner ursprünglichen Zielsetzung ist der Raspberry Pi vielfältig einsetzbar. So existiert ein großes Zubehörangebot für zahlreiche Anwendungsbereiche, wie beispielsweise Media Center, Robotersteuerung oder Thin Clients.

Das aktuelle Modell Raspberry Pi 3 B+ (Abbildung 3) hat einen 1,4GHz 64-Bit-Vierkernprozessor, 1GB RAM, Gigabit Ethernet, Wireless LAN, Bluetooth, einen HDMI-Anschluss und vier USB 2.0 Ports. Des weiteren verfügt er über eine frei programmierbare 40-Pin GPIO-Schnittstelle (General Purpose Input/Output), über welche zahlreiche elektronische Bauteile angesteuert werden können, wie beispielsweise LEDs oder Sensoren. Die GPIO-Pins können teilweise als SPI-, I2C- oder UART-Schnittstelle verwendet werden.

Abbildung 3: Raspberry Pi 3 Model B+ (Quelle: [8])

Als Betriebssystem können auf dem Raspberry Pi verschiedene Linux-Distributionen eingesetzt werden aber auch eine spezielle Version von Windows 10 ist verfügbar. Das am meisten verbreitete und von der Raspberry Pi Foundation empfohlene System ist Raspbian, welches auf Debian GNU/Linux basiert. Der Raspberry Pi bootet das Betriebssystem von einer wechselbaren SD-Speicherkarte oder bei neueren Modellen auch von einem USB-Speichergerät.

2.4 Pigator

Der Pigator [9] ist ein Modulträger für den Raspberry Pi, welcher von Busware entwickelt wird. Er wird auf die GPIO-Schnittstelle des Raspberry Pi gesteckt und kann anschließend um eines von vielen verschiedenen Pigator-Modulen erweitert werden. Unter anderem gibt es ein KNX-Modul (Abbildung 4) mit welchem der Raspberry Pi an einen KNX-Bus angebunden werden kann. Der Träger und das zugehörige Modul kosten jeweils etwa 40 EUR.

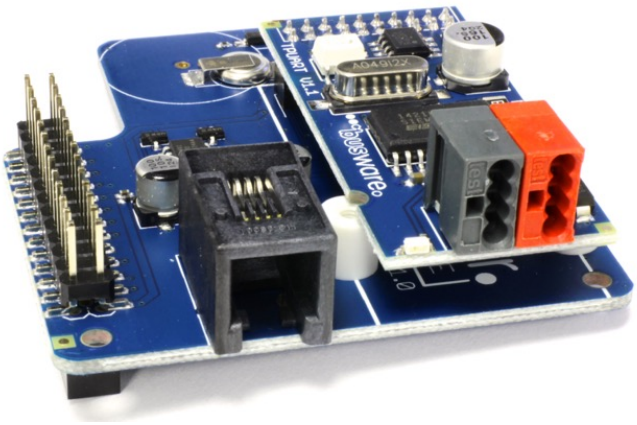

Abbildung 4: Pigator mit KNX-Modul (Quelle: [9])

Mit knxd [10] steht eine unter Linux (und somit auf dem Raspberry Pi) lauffähige Software zur Verfügung, welche das Pigator KNX-Modul unterstützt und so eine Schnittstelle zum KNX-Bus bietet. Damit lässt sich ein KNX/IP-Gateway bauen, das eine preiswerte und flexiblere Alternative zu den von der KNX Association zertifizierten KNX/IP-Gateways darstellt.

3 Anforderungen

Nachfolgend sollen die Anforderungen an das geplante System und den darin enthaltenen Schlüsselspeicher vorgestellt werden. Dabei wird sowohl auf funktionale und monetäre als auch auf die Sicherheit betreffende Aspekte eingegangen.

3.1 Funktionale Anforderungen

Das Raspberry Pi-basierte Intrusion Detection System zeichnet KNX Data Secure-Pakete auf dem Bus auf, untersucht diese und versucht so Angriffsmuster zu erkennen. Bevor die Pakete weiter analysiert werden können, müssen diese erst entschlüsselt und deren Integrität durch MAC-Berechnung geprüft werden. Das ganze erfolgt über AES-128 CCM und die zugehörigen Schlüssel müssen zugreifbar hinterlegt sein.

Das betrachtete System soll in der Lage sein bis zu 100 KNX Data Secure-Laufzeitschlüssel persistent zu speichern. Diese Anzahl ist für eine mittelgroße KNX-Installation mit Data Secure-Geräten durchaus im normalen Bereich, da für verschiedene Geräte und auch deren Gruppen jeweils unterschiedliche Schlüssel verwendet werden.

Weiterhin wäre eine lückenlose Überwachung der Pakete wünschenswert, ohne das einzelne Pakete verworfen werden müssen. Dazu ist es notwendig, dass die Geschwindigkeit der Entschlüsselung und MAC-Bestimmung für die Datenmenge auf dem KNX-Bus ausreichend ist.

Eine zusätzliche Anforderung an den Schlüsselspeicher ergibt sich daraus, dass dieser möglichst portabel sein sollte. Beispielsweise könnte der Schlüsselspeicher nach einem erfolgten Angriff unter Ausnutzung einer Schwachstelle im Betriebssystem an an einem „sauberen“ Raspberry Pi weiter verwendet werden. Das gilt natürlich nur unter der Voraussetzung, dass keine Schlüssel kopiert wurden.

Eine weitere aber durchaus kritisch abzuwegende Anforderung stellt die direkte Verfügbarkeit des IDS nach einem Neustart des Systems dar. Dazu ist es notwendig, dass der Schlüsselspeicher ohne Benutzerinteraktion zugreifbar ist, was in den meisten Fällen aber bedeuten würde, dass ein zugehöriges Passwort oder eine PIN im System hinterlegt werden. Dieses Szenario ergibt sich beispielsweise nach einem Stromausfall.

3.2 Anforderungen an die Kosten

Die Kosten für mögliches Hardwarezubehör für den Schlüsselspeicher sollen sich im Rahmen des Raspberry Pi und der Pigator-Module bewegen. In der Summe ist für das Gesamtsystem ein Preis im niedrigen dreistelligen Eurobereich angedacht.

3.3 Sicherheitsanforderungen

3.3.1 Schutzziele

In einem Informationssystem können verschiedene Schutzziele [11] umgesetzt werden. Nachfolgend sollen die für den vorliegenden Anwendungsfall relevanten Ziele aufgezeigt werden.

Authentizität Nur authentifizierte Benutzer sollen Zugriff auf die Entschlüsselungs- und MAC-Berechnungsfunktionen und somit indirekt auf die entsprechenden Laufzeitschlüssel haben.

Integrität Die gespeicherten Laufzeitschlüssel sollen nicht unautorisiert und unbemerkt veränderbar sein.

Vertraulichkeit Es darf kein unautorisierter Lesezugriff auf die gespeicherten Laufzeitschlüssel erfolgen.

Verfügbarkeit Ein autorisierter Zugriff auf die Entschlüsselungs- und MAC-Berechnungsfunktionen und somit indirekt auf die entsprechenden Laufzeitschlüssel soll nicht beeinträchtigt sein.

3.3.2 Angreifermodell

Ein Angriff auf das IDS-System und den zugrunde liegenden Schlüsselspeicher kann lohnenswert sein, da er einen nachfolgenden Angriff auf die gesamte KNX-Installation ermöglicht. KNX-Systeme wie Torsteuerungen oder Alarmanlagen machen solche Angriffe auch für Kriminelle wie beispielsweise Einbrecher interessant.

Im definierten Szenario soll ein potenzieller Angreifer sowohl Zugriff übers Netz als auch physischen Zugriff auf das IDS-System haben. Beim Netz-Zugriff (KNX-Bus und Ethernet) kann er beliebige Datenpakete abgreifen aber auch eigene Pakete senden. Durch

physischen Zugriff auf den Raspberry Pi hat er eine Reihe von Möglichkeiten, wie etwa das Kopieren oder Verändern der SD-Karte oder auch das Manipulieren der Hardware.

Ein Angreifer kann bekannte Schwachstellen wie Buffer-Offerflows im Betriebssystem oder in den Diensten des Raspberry Pi ausnutzen. Damit erlangt er möglicherweise Root-Rechte und hat somit volle Systemkontrolle. Auch kann er dann Malware wie etwa Keylogger auf dem System installieren um damit Tastatureingaben zu protokollieren.

3.3.3 Bedrohungsanalyse

Durch mögliche Ausnutzung von Schwachstellen kann das System verschiedenen Bedrohungen ausgesetzt sein, welche unterschiedlich schwere Gefährdungen darstellen.

Missbrauch der Entschlüsselungs- bzw. MAC-Berechnungsfunktion Der Angreifer erlangt Zugriff auf die Entschlüsselungs- und MAC-Berechnungsfunktionen und kann diese für eigene Zwecke missbrauchen. Er kommt nicht in den Besitz der Laufzeitschlüssel.

Verändern/Löschen der Laufzeitschlüssel Dem Angreifer gelingt es die gespeicherten Schlüssel zu verändern oder zu löschen. Er kommt nicht in den Besitz der Laufzeitschlüssel. Das kann zu einem Betriebsausfall (Denial of Service) des IDS-Systems führen.

Kopieren der Laufzeitschlüssel Der Angreifer kommt in den Besitz (aller) Laufzeitschlüssel. Diese Bedrohung ist sehr schwerwiegend und entsprechende Angriffe sollten unbedingt verhindert werden. Das Erneuern von 100 Schlüsseln ist sehr aufwendig und kann sogar zum temporären Betriebsausfall der gesamtem KNX-Installation führen.

3.4 Annahmen und Abgrenzung

Der Import der Laufzeitschlüssel erfolgt vor dem Start des IDS-Systems in einem gesicherten Umfeld. In dieser Arbeit wird das Vorhandensein der Schlüssel im jeweiligen Speicher als gegeben angenommen und nicht weiter betrachtet. Natürlich müsste bei einer tatsächlichen Implementation des Systems hierauf eingegangen werden.

Die Sicherheitsbetrachtungen dieser Arbeit beschränken sich auf die oben genannten Punkte bezüglich der Schlüsselspeicherung, wobei die Sicherung der Vertraulichkeit der Laufzeitschlüssel im Vordergrund steht. Die Sicherheit von KNX Data Secure, AES,

dem Raspberry Pi im Allgemeinen und der tatsächlichen IDS-Software sind dabei out-of-scope.

4 Vergleich der Speicherlösungen

Im folgenden werden verschiedene Speichermöglichkeiten für KNX-Laufzeitschlüssel vorgestellt. Dabei wird die jeweilige Lösung in Bezug auf die oben besprochenen Anforderungen (Funktionalität, Kosten und Sicherheit) analysiert. Auch wenn der Vergleich eher konzeptioneller Natur ist, so wird an einigen Stellen auch kurz auf konkrete Implementationen eingegangen.

4.1 Speicherung im Dateisystem

Die Laufzeitschlüssel können im Dateisystem der im Raspberry Pi eingelegten SD-Karte gespeichert werden. Die Implementation sollte nicht weiter schwer sein, da die Schlüssel beispielsweise in einer einfachen Textdatei abgelegt werden können.

Da SD-Karten heutzutage 256GB und mehr Speicherkapazität haben, stellt eine Speicherung von 100 und mehr 128-Bit-AES-Schlüsseln kein Problem dar. Auch ist der Prozessor des Raspberry Pi 3 B+ leistungsfähig genug, um die Entschlüsselung und MAC-Berechnung ausreichend schnell ausführen zu können. Was die Portabilität des Schlüsselspeichers angeht, so ist dieser zwar auf derselben SD-Karte auf der sich auch das Betriebssystem des Raspberry Pi befindet, die Datei mit den Schlüsseln kann aber einfach auf eine weitere „saubere“ SD-Karte kopiert werden. Nach einem Neustart des Systems kann dieses ohne weitere Nutzerinteraktionen direkt auf die Schlüssel zugreifen, da diese nicht durch ein Passwort oder eine PIN geschützt sind. Da für die Speicherung der Schlüssel keine extra Hardware benötigt wird, fallen auch keine weiteren Kosten an.

Zwar können die Schlüssel eingeschränkten Zugriffsrechten im Linux-Dateisystem unterliegen, allerdings kann beispielsweise Malware mit Root-Zugriff diese trotzdem auslesen oder überschreiben. Auch kann die SD-Karte mit den gespeicherten Schlüsseln bei physischem Zugriff auf den Raspberry Pi leicht kopiert werden. Hat der Angreifer erst mal die Schlüssel, kann er anschließend sämtliche KNX-Telegramme ent- oder verschlüsseln und auch problemlos MACs berechnen.

Das Speichern der Schlüssel im Dateisystem ist für das geplante Szenario nicht geeignet. Zwar erfüllt die Lösung alle funktionellen Anforderungen, ist aber aus Sicherheitssicht nicht brauchbar.

4.2 Speicherung im Dateisystem verschlüsselt mit Passwort

Die Speicherung der Laufzeitschlüssel erfolgt ähnlich wie im letzten Abschnitt im Dateisystem, nur dass die Schlüssel zusätzlich noch mit einem Passwort geschützt werden. Dazu werden die Schlüssel, bevor sie ins Dateisystem geschrieben werden, mit einem symmetrischen Kryptoalgorithmus wie beispielsweise AES verschlüsselt. Der dazu nötige symmetrische Schlüssel wird mit Hilfe eines kryptographischen Schlüsselableitungsverfahrens aus dem Passwort abgeleitet. Ein Beispiel für ein solches Schlüsselableitungsverfahren ist PBKDF2, welches im PKCS #5 (RFC 8018 [12]) beschrieben ist.

Die funktionellen Anforderungen werden bis auf eine Ausnahme wie oben beschrieben erfüllt. Falls die direkte Verfügbarkeit der Schlüssel ohne Nutzerinteraktion nach einem Neustart gewünscht wird, muss das zugehörige Passwort in einer Datei hinterlegt werden. Damit wird das Sicherheitsniveau allerdings auf das der Lösung des vorherigen Abschnitts gesenkt.

Der Vorteil der verschlüsselten Speicherung besteht darin, das einfaches Kopieren der SD-Karte nicht direkt zum Besitz der Schlüssel führt. Falls das Passwort jedoch nicht in einer Datei hinterlegt wird und es vom Nutzer beim Neustart des Systems jedes Mal eingegeben werden muss, kann ein Angreifer dieses Passwort mit einem Keylogger ermitteln. Auch kann er versuchen, das Passwort durch einen Brute-Force-Angriff herauszubekommen. Wenn der Angreifer in den Besitz des Passwortes gelangt, dann kann er die Laufzeitschlüssel lesen und auch verändern. Ein weiteres Problem liegt darin, das die Schlüssel zur Laufzeit im Speicher des Raspberry Pi im Klartext vorliegen. Damit kann eine Malware auch ohne Besitz des Passwortes diese auslesen.

Auch wenn die passwortverschlüsselte Speicherung im Dateisystem eine zusätzliche Sicherheitsschicht einfügt, so ist es Angreifern trotzdem möglich in den Besitz der Laufzeitschlüssel zu gelangen. Somit ist auch diese Lösung nicht geeignet.

4.3 Speicherung auf einer Smartcard

Smartcards [13] sind Plastikkarten, die einen Prozessor, einen ROM-Speicher mit Kartenbetriebssystem und nichtflüchtigen EEPROM-Speicher enthalten. Ihre wesentlichen Eigenschaften werden in der ISO-Norm 7816 festgelegt. Neben den weit verbreiteten Karten im Kreditkartenformat, gibt es auch andere Formfaktoren wie etwa USB-Token. Smartcards besitzen kryptografische Fähigkeiten und auf ihnen gespeicherte Daten sind gegen unerwünschte Zugriffe und gegen Manipulation geschützt. So ist es möglich, geheime Daten in die Karte zu laden, welche anschließend nicht mehr von außen gelesen werden können. Auch können Daten und Kommandos der Smartcard durch eine PIN geschützt werden. Moderne Smartcards sind frei programmierbar und damit in ihrer

Anwendung sehr flexibel.

Eine für das angedachte Szenario passende Smartcard ist beispielsweise der CryptoMate Nano (Abbildung 5) von Advanced Card Systems [14]. Dabei handelt es sich um ein ISO 7816-konformes kryptografisches USB-Token mit 64kB EEPROM. Der gewählte Formfaktor ist im Kontext des Raspberry Pi passend und auch preislich (ca. 20 EUR) bewegt sich das Token im gewünschten Rahmen. Der CryptoMate Nano unterstützt AES-128 im ECB- und CBC-Betriebsmodus, wobei der für KNX Data Secure benötigte Counter Mode durch den ECB Mode nachgebildet werden kann.

Abbildung 5: CryptoMate Nano USB-Token (Quelle: [14])

Die Anzahl der im EEPROM einer Smartcard gespeicherten Schlüssel hängt von dessen Kapazität ab. Beispielsweise kann man in einem 64kB EEPROM theoretisch 4096 AES-128-Schlüssel speichern. Praktisch ist der Wert um einiges kleiner, da im EEPROM auch Metadaten zu den Schlüsseln und andere Daten wie etwa PINs abgelegt werden. Trotzdem sollte es möglich sein, 100 Laufzeitschlüssel zu speichern. Der Prozessor bzw. Koprozessor einer aktuellen Smartcard ist leistungsfähig genug, um die AES-Kryptooperationen in benötigter Geschwindigkeit auszuführen. In wie weit die USB-Schnittstelle ein Bottleneck darstellt, müsste noch konkret untersucht werden, es ist aber davon auszugehen, dass sie für den vorliegenden KNX-Fall ausreichend schnell ist. Smartcards im USB-Token-Formfaktor können problemlos von einem Raspberry Pi an einen anderen gesteckt werden. Wie auch im vorhergehenden Abschnitt beschrieben, müsste für die direkte Verfügbarkeit des Systems nach einem Neustart die PIN für den Zugriff auf die Kryptofunktionen in einer Datei hinterlegt werden, was einen negativen Einfluss auf das Sicherheitsniveau hat. Die Kosten für geeignete Smartcards bewegen sich im ein- bis niedrigen zweistelligen Eurobereich.

Der Zugriff auf die Entschlüsselung und MAC-Berechnung sollte erst nach einer PIN-Eingabe möglich sein. Diese könnte zwar durch einen Keylogger abgegriffen werden, allerdings hat der Angreifer damit trotzdem nicht die Laufzeitschlüssel, welche unter der

Annahme einer sicheren Smartcard nicht aus der Karte ausgelesen werden können. Brute-Force-Angriffe auf die PIN lassen sich durch einen Fehleingabezähler verhindern. Das Verändern oder Löschen der Schlüssel kann durch eine weitere (Admin)-PIN abgesichert werden, welche zur Laufzeit des Systems niemals eingegeben wird. Ein Angreifer mit physischem Zugriff auf den Raspberry Pi kann das Token mitnehmen und dadurch das System unbrauchbar machen, was aber für alle Speicherlösungen gilt. Auch hier hat er aber keinen direkten Zugriff auf die Schlüssel, da die Smartcard vor entsprechenden Manipulationen geschützt ist.

Das Speichern der Laufzeitschlüssel in einer Smartcard stellt einen geeigneten Ansatz dar. Es werden sowohl die funktionellen als auch die Sicherheitsanforderungen größtenteils erfüllt. Besonders ist hervorzuheben, dass die Schlüssel gegen unerwünschtes Kopieren geschützt sind.

4.4 Logische Speicherung auf einer Smartcard

Falls mehr Laufzeitschlüssel in einer Smartcard gespeichert werden sollen, als in deren EEPROM-Speicher passen, kann ein hybrider Ansatz zur Speicherung gewählt werden. Dabei werden die Schlüssel auf der Karte verschlüsselt und dann in verschlüsselter Form ins Dateisystem der SD-Karte ausgelagert. Vor Benutzung dieser Schlüssel müssen diese wieder in die Smartcard importiert und intern entschlüsselt werden. Eine zu wählende Verdrängungsstrategie entscheidet darüber, welche Schlüssel beim Import überschrieben werden. Beispiele für solche Algorithmen sind LFU (Least Frequently Used) oder auch LRU (Least Recently Used). Für die Umsetzung dieser Lösung sind Smartcards mit einem festen Befehlssatz unter Umständen nicht ausreichend. Hierfür sind frei programmierbare Karten wie die Java Card geeignet, in denen die gewünschte Funktionalität implementiert werden kann.

Verglichen mit der reinen Smartcard-Lösung aus dem letzten Abschnitt, ist die Umsetzung der Implementation um einiges komplexer. Auch ist die Portabilität des Schlüsselspeichers leicht eingeschränkt, da neben dem USB-Token, auch die Schlüssel aus dem Dateisystem benötigt werden. Außerdem ist die Geschwindigkeit etwas geringer, da der Schlüsselimport und die Entschlüsselung dieser auch Zeit benötigen. Ein Vorteil ist, dass der Speicherplatz für die Schlüssel erhöht ist, da die SD-Karte eine weit höhere Kapazität als der EEPROM der Smartcard hat.

Ein Schutz vor Änderung der Schlüssel in der Smartcard durch eine Admin-PIN ist nicht mehr gegeben, da die Schlüssel im laufenden Betrieb immer wieder überschrieben werden müssen. Außerdem kann die Integrität der Schlüssel im Dateisystem der SD-Karte nicht garantiert werden, da sie ein Angreifer mit System-Zugriff problemlos ändern könnte. Das kann durch MAC-Berechnung für die Schlüssel vermieden werden, erhöht aber wiederum den Aufwand bei der Implementation. Des Weiteren ergibt sich ein zusätzli-

cher Angriffsvektor daraus, dass Schlüssel im Dateisystem der SD-Karte liegen und ein Angreifer versuchen könnte deren Verschlüsselung (offline) zu brechen.

Die hybride Lösung für die Speicherung der Laufzeitschlüssel führt zu zusätzlichem Aufwand bei der Implementation und hat ein niedrigeres Sicherheitsniveau als eine reine Smartcard-Lösung. Wenn möglich sollte stattdessen eine Smartcard mit größerem EEPROM benutzt werden. Falls das nicht möglich ist oder aber eine sehr große Anzahl von Schlüsseln gespeichert werden soll, stellt der hybride Ansatz eine brauchbare Alternative dar.

4.5 Speicherung in einem HSM

Ein Hardware-Sicherheitsmodul (HSM) [15] ist ein internes (PCI Express) oder externes (Netzwerkanbindung) Peripheriegerät, welches die effiziente und sichere Ausführung kryptographischer Operationen ermöglicht. HSMs bieten einen umfangreichen Schutz der kryptographischen Schlüssel gegen verschiedene Arten von Angriffen und besitzen Funktionen wie etwa Zugriffsschutz im Mehr-Augen-Prinzip oder verschlüsseltes Backup der Schlüssel. Je nach Höhe des Sicherheitsniveaus werden HSMs nach verschiedenen Sicherheitsstandards zertifiziert.

HSMs liegen funktions- und sicherheitstechnisch im oberen Bereich und würden die geplante Aufgabe problemlos erfüllen. Allerdings ist deren hard- und softwareseitige Anbindung an den Raspberry Pi nicht so einfach und die Zielstellung der Speicherung der Schlüssel auf einem Raspberry Pi würde damit aus den Augen verloren. Außerdem bewegt sich der Preis für ein solches Gerät im bis zu fünfstelligen Eurobereich, was preislich weit außerhalb des Rahmens liegt. Aus diesen Gründen werden HSMs nachfolgend nicht weiter betrachtet und es soll stattdessen eine Art „HSM light“ für den Raspberry Pi vorgestellt werden.

Der Zymkey (Abbildung 6) von der Firma Zymbit [16] ist ein Sicherheitsmodul für den Raspberry Pi. Die aktuell erhältliche Version 5 beherrscht neben anderen Kryptoalgorithmen auch AES und hat einen manipulationssicheren Schlüsselspeicher für 32 Schlüssel. Der Zymkey kann unter anderem über APIs für C, C++ und Python angesprochen werden. Der Preis für das Modul bewegt sich im Bereich von 50 USD.

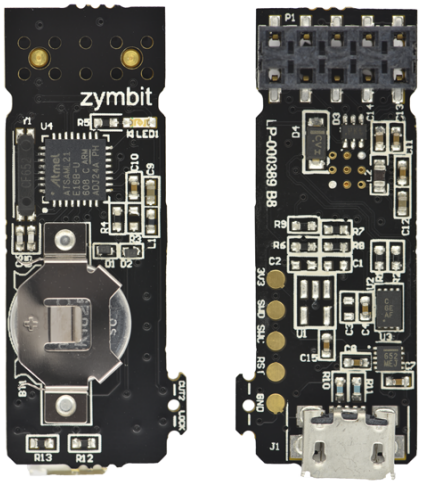

Abbildung 6: Zymkey für den Raspberry Pi (Quelle: [16])

Der Zymkey wird auf die GPIO-Schnittstelle des Raspberry Pi gesteckt und über den I2C-Bus angesprochen. Eventuell kann es zu einem Konflikt mit dem Pigator-Modul kommen, welches ebenfalls über GPIO angebunden wird. Alternativ könnte man hier die USB-Variante des Pigator nutzen um das Problem zu umgehen. Die Geschwindigkeit des zugrunde liegenden Kryptoprozessors und der I2C-Anbindung müssten im Detail untersucht werden, um genauere Aussagen darüber treffen zu können. Der Zymkey muss vor der Benutzung an einen spezifischen Raspberry Pi gebunden werden, da sonst manche Funktionen wie beispielsweise die Perimetererkennung nicht nutzbar sind. Diese Bindung ist, sobald sie einmal erfolgt ist, nicht mehr umkehrbar. Einerseits ist das ein schönes Sicherheitsfeature, andererseits verhindert das die Portabilität des Moduls.

Das Sicherheitsniveau des Zymkeys ist in etwa mit dem einer Smartcard vergleichbar. Damit gilt das bezüglich der Sicherheit im Smartcard-Abschnitt Gesagte auch hier.

Der Zymkey bietet für das geplante Szenario keinen relevanten Mehrwert gegenüber einer Smartcard. Demgegenüber stehen der leicht höhere Preis, der kleinere Schlüsselspeicher und eventuelle Probleme bei der Kompatibilität mit dem Pigator und der Portabilität. Die Smartcard scheint hier also die bessere Lösung zu sein. Nichtsdestotrotz hat der Zymkey seine Berechtigung und stellt eine mögliche Alternative zu Smartcard dar.

4.6 Speicherung in einem TPM

Ein Trusted Platform Module (TPM) [17][11] ist ein Chip, welcher der Spezifikation der Trusted Computing Group [18] folgt und grundlegende kryptografische Dienste implementiert. Im Vergleich zu einer Smartcard ist ein TPM an eine konkrete Hardware-Instanz gebunden, während eine Smartcard einer Benutzer-Instanz zugeordnet wird. Aufgrund eines spezifischen kryptografischen Schlüssels im TPM ist es möglich, die Hardware-Instanz eineindeutig zu identifizieren.

Die Speicherressourcen eines TPM sind beschränkt, wobei es nur etwa fünf bis zehn Schlüssel-Speicherslots [19] hat. Allerdings ist es durch Schlüssel-Wrapping möglich, verschlüsselte Schlüssel außerhalb des TPM abzulegen, was es ermöglicht, eine nahezu unbegrenzte Anzahl an Schlüsseln zu verwalten. Obwohl ein einzelnes TPM fest an eine Hardware gebunden ist, können einzelne Schlüssel als migrierbar markiert und so auf ein anderes TPM übertragen werden.

Das Sicherheitsniveau von TPMs entspricht in etwa dem von Smartcards. So können TPMs auch Funktionen wie etwa Absicherung gegen physische Manipulation oder Zugriffskontrolle durch Passwörter bieten. Bezüglich der definierten Sicherheitsanforderungen ist ein TPM aufgrund des Schlüssel-Wrappings vergleichbar mit der hybriden Speichermethode auf einer Smartcard.

Ein mögliches TPM für den Raspberry Pi stellt das LetsTrust TPM (Abbildung 7) von pi3g [20] dar. Dabei handelt es sich um ein GPIO-Aufsteckmodul, welches per SPI-Schnittstelle mit dem Raspberry Pi kommunizieren kann. Das LetsTrust TPM kostet 25 EUR und ist mit der TPM-Spezifikation 2.0 konform. Um es unter Linux lauffähig zu machen, wird ein gepatchter Kernel benötigt.

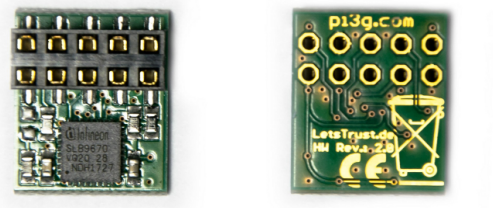

Abbildung 7: LetsTrust TPM für den Raspberry Pi (Quelle: [20])

Durch die Anbindung des LetsTrust TPM über SPI kann davon ausgegangen werden, dass die Übertragungsgeschwindigkeit nicht sehr hoch ist. Des Weiteren wird das Schlüssel-Wrapping über RSA realisiert, was wahrscheinlich auch zu Performanceeinbu-

ßen führt. Weiterhin müsste aufgrund der GPIO-Schnittstelle im Detail geklärt werden, inwieweit es zu Konflikten mit dem Pigator-Modul kommen kann. Der LetsTrust TPM basiert auf dem Infineon SLB 9670VQ2.0, welcher laut Spezifikation [21] kein AES implementiert. Damit scheidet er leider für unseren geplanten Einsatzzweck aus.

Auch wenn der LetsTrust TPM für den konkreten Zweck nicht geeignet ist, so stellen Trusted Platform Module aus konzeptioneller Sicht trotzdem eine weitere interessante Alternative zur Smartcard dar. So könnte ein performantes, AES-fähiges TPM als Speicher für die Laufzeitschlüssel genutzt werden.

4.7 Speicherung in der ARM TrustZone

Die ARM TrustZone [22] ist eine Technologie für ARM-Prozessoren zur Realisierung eines Trusted Execution Environment (TEE) [23]. Ein TEE ist eine isolierte Laufzeitumgebung, welche die Vertraulichkeit und Integrität der darin enthaltenen Daten und laufenden Applikationen sicherstellt. Dabei handelt es sich um eine systemweite Hardwareisolation welche auch Peripheriegeräte umfassen kann. Um die Standardisierung des TEE-Konzepts kümmert sich der Global Platform-Verband.

Es existieren zahlreiche TEE-Implementationen, sowohl kommerzielle als auch Open Source-Varianten. Dabei handelt es sich um eine Art vertrauenswürdiges Betriebssystem welches parallel zum Standardbetriebssystem läuft. Ein Beispiel für ein solches System für die ARM TrustZone ist OP-TEE (Open Portable Trusted Execution Environment), welches von der Linaro-Initiative weiterentwickelt wird und auch auf den Raspberry Pi 3 portiert wurde [24].

OP-TEE bietet die Möglichkeit Schlüssel vertrauenswürdig und persistent in einem Standard-Dateisystem zu speichern [25]. Dabei werden die Schlüssel vorher mit einem File Encryption Key (FEK) verschlüsselt, welcher für jede abgespeicherte Datei individuell erzeugt wird. Der FEK wiederum wird mit einem Trusted Application Storage Key (TSK) verschlüsselt, der in einem mehrstufigen Prozess abgeleitet wird, wobei auch spezifische Attribute der jeweiligen Hardware-Instanz mit einfließen. Daraus ergibt sich, dass die gespeicherten Schlüssel an die konkrete Hardware-Instanz gebunden sind. Mit dieser Methode ist es somit möglich eine sehr große Anzahl an Laufzeitschlüsseln zu speichern.

Verglichen mit einer Smartcard oder einem TPM ist das TEE performanter, da der Prozessor und Hauptspeicher des Systems genutzt werden. Außerdem bieten die Applikationen im TEE eine höhere Funktionalität, da sie flexibel programmiert werden können.

Das Sicherheitsniveau von TEEs ist ähnlich dem der hybriden Speicherung auf Smart-

cards oder im TPM, da die Laufzeitschlüssel auch hier gesichert im Dateisystem abgelegt werden. Im Vergleich zu den bisher vorgestellten Lösungen, ist es durch ein TEE allerdings möglich, dem Angreifer das Ausspähen eines Passworts oder einer PIN zu erschweren. Gegen Hardware-Keylogger hilft das natürlich nichts. Leider bietet der ARM-Prozessor des Raspberry Pi 3 keine vollständige ARM TrustZone-Implementation und der OP-TEE-Port dient nur Bildungs- oder Testzwecken [24]. Dadurch ist beispielsweise der Speicherbereich für vertrauenswürdige Applikationen nicht vollständig vom restlichen Speicher isoliert. Das macht die ARM TrustZone des Raspberry Pi für das gewünschte Vorhaben unbrauchbar.

Wie auch schon beim TPM ist die ARM TrustZone konzeptionell sehr interessant, aber in ihrer Umsetzung für den Raspberry Pi unzureichend. So erlaubt es ein Trusted Execution Environment die Passwort- oder PIN-Eingabe teilweise abzusichern, was verglichen mit den anderen Ansätzen ein Vorteil ist. Auch könnte durch die Kombination einer vollumfänglich funktionierenden ARM TrustZone mit beispielsweise einer Smartcard das Sicherheitsniveau der Schlüsselspeicherung nochmals erhöht werden.

5 Ausblick

Der nächste Schritt wäre die Umsetzung eines Prototypen. Dazu müsste eine Softwarekomponente implementiert werden, welche KNX Data Secure-Telegramme übergeben bekommt, diese entsprechend unter Zuhilfenahme des Schlüsselspeichers entschlüsselt und verifiziert und anschließend an eine potenzielle IDS-Software weitergibt. Bei der Realisierung von aufwändigeren Varianten, wie der hybriden Schlüsselspeicherung in einer Smartcard, müsste voraussichtlich auch eine entsprechende Applikation für die Smartcard umgesetzt werden.

Ein weiterer wichtiger Punkt ist der Import der Laufzeitschlüssel in den Schlüsselspeicher. Wie schon bereits erwähnt, müsste hierfür ein schlüssiges Konzept gefunden werden. Eine Möglichkeit wäre eventuell die Anbindung des Schlüsselspeichers direkt an die ETS.

Auch könnte die Schlüsseldiversifizierung betrachtet werden, welche eine Speicher sparende Alternative darstellt. Statt einzelne Schlüssel zu speichern, würden Regeln zu deren Erzeugung hinterlegt werden. Die Laufzeitschlüssel können dann bei Bedarf über eine Schlüsselableitungsfunktion von einem Masterschlüssel abgeleitet werden. Hierfür müsste allerdings geklärt werden, inwieweit beispielsweise Smartcards und auch KNX Data Secure (und hier vor allem die ETS) das unterstützen.

Da KNX Data Secure-fähige Geräte heutzutage immer noch kaum anzutreffen sind, könnte überlegt werden, die in dieser Arbeit vorgestellten Lösungen auch für andere Szenarien neben KNX umzusetzen.

6 Fazit

In dieser Arbeit wurden verschiedene Möglichkeiten der Speicherung geheimer Schlüssel auf einem Raspberry Pi vor dem Hintergrund eines KNX Data Secure-Intrusion Detection Systems untersucht. Die verglichenen Lösungen unterscheiden sich dabei teils stark in ihren funktionalen, preislichen und besonders auch sicherheitsrelevanten Eigenschaften und sind so unterschiedlich gut für den geplanten Einsatzzweck geeignet.

Als am geeignetsten stellte sich dabei die Speicherung der Schlüssel auf einer Smartcard heraus. Der EEPROM-Speicher der Karte ist normalerweise groß genug für die benötigte Anzahl von Schlüsseln und auch preislich bewegt sich eine Smartcard in bezahlbaren Regionen. Das wichtigste ist allerdings, dass auf einer Smartcard gespeicherte Schlüssel gegen unerwünschte Zugriffe und gegen Manipulation geschützt werden können und so ein Angreifer nicht ohne Weiteres in deren Besitz gelangen kann.

7 Literatur

[1] Wikipedia-Autoren: *KNX-Standard – Wikipedia*. https://de.wikipedia.org/wiki/KNX-Standard, besucht: 2018-09-10.

[2] KNX Association: *Offizielle Webseite der KNX Association*. https://www.knx.org, besucht: 2018-09-10.

[3] Merten: *Wie funktioniert KNX*. https://www.merten.de/Wie-funktioniert-KNX.1347.0.html, besucht: 2018-09-10.

[4] KNX Association: *ETS5 Professional*. https://www.knx.org/knx-en/for-professionals/software/ets-5-professional/index.php, besucht: 2018-09-10.

[5] KNX Association: *Application Note 158 – KNX Data Security*, 2013.

[6] KNX Association: *KNX Sicherheit – KNX Positionspapier zu Datensicherheit und Datenschutz*, 2014.

[7] Wikipedia-Autoren: *Raspberry Pi – Wikipedia*. https://de.wikipedia.org/wiki/Raspberry_Pi, besucht: 2018-09-12.

[8] Raspberry Pi Foundation: *Webseite der Raspberry Pi Foundation*. https://www.raspberrypi.org/, besucht: 2018-09-12.

[9] Busware: *Pigator*. http://busware.de/tiki-index.php?page=PIGATOR, besucht: 2018-09-12.

[10] Matthias Urlichs: *Pigator – GitHub*. https://github.com/knxd/knxd, besucht: 2018-09-12.

[11] Claudia Eckert: *IT-Sicherheit: Konzepte – Verfahren – Protokolle*. Oldenbourg, München, 9. Auflage, 2014.

[12] K. Moriarty, B. Kaliski und A. Rusch: *PKCS #5: Password-Based Cryptography Specification – Version 2.1*. RFC 8018, RFC Editor, 2017. https://www.rfc-editor.org/rfc/rfc8018.txt.

[13] Wolfgang Rankl und Wolfgang Effing: *Handbuch der Chipkarten – Aufbau, Funktionsweise, Einsatz von Smart Cards*. Hanser, München, 4. Auflage, 2002.

[14] Advanced Card Systems Ltd.: *CryptoMate Nano – Cryptographic USB Token.* https://www.acs.com.hk/en/products/414/cryptomate-nano-cryptographic-usb-tokens/, besucht: 2018-09-08.

[15] Wikipedia-Autoren: *Hardware-Sicherheitsmodul – Wikipedia.* https://de.wikipedia.org/wiki/Hardware-Sicherheitsmodul, besucht: 2018-09-17.

[16] Zymbit: *Zymkey – Security Module for Raspberry Pi.* https://www.zymbit.com/zymkey/, besucht: 2018-09-17.

[17] Wikipedia-Autoren: *Trusted Platform Module – Wikipedia.* https://de.wikipedia.org/wiki/Trusted_Platform_Module, besucht: 2018-09-18.

[18] Trusted Computing Group: *Trusted Platform Module Library Specification, Family 2.0, Level 00, Revision 01.16*, 2014.

[19] Will Arthur, David Challener und Kenneth Goldmann: *A Practical Guide to TPM 2.0: Using the Trusted Platform Module in the New Age of Security.* Apress, 2015.

[20] buyzero: *LetsTrust TPM für den Raspberry Pi.* https://buyzero.de/products/letstrust-hardware-tpm-trusted-platform-module, besucht: 2018-09-18.

[21] Infineon Technologies: *Infineon SLB 9670VQ2.0.* https://www.infineon.com/cms/de/product/security-smart-card-solutions/optiga-embedded-security-solutions/optiga-tpm/slb-9670vq2.0/, besucht: 2018-09-18.

[22] Arm Developer: *Arm TrustZone.* https://developer.arm.com/technologies/trustzone, besucht: 2018-09-18.

[23] Wikipedia-Autoren: *Trusted Execution Environment – Wikipedia.* https://de.wikipedia.org/wiki/Trusted_Execution_Environment, besucht: 2018-09-18.

[24] Linaro-Initiative: *OP-TEE on Raspberry Pi 3.* https://www.op-tee.org/docs/rpi3/, besucht: 2018-09-18.

[25] Linaro-Initiative: *Secure Storage in OP-TEE – GitHub.* https://github.com/OP-TEE/optee_os/blob/master/documentation/secure_storage.md, besucht: 2018-09-18.