ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

Implementierung eines sicheren Heimnetzes für IoT-Anwendungen

Dokumentation einer Projektarbeit

Julian Krieger Yasin Tatar

30. August 2019

Inhaltsverzeichnis

1. Einführung

1.1. Ausgangslage

Gegenwärtig steigt die Anzahl an vernetzten Geräten in der Industrie, aber auch in Haushalten, stark an [Sta16]. Maßnahmen zum Schutz dieser Geräte sind jedoch oft unzureichend, oder nicht vorhanden [ONe16]. Gerade der sog. „Smart Home“-Bereich ist besonders betroffen, da die Geräte potenziell einen direkten, physischen oder virtuellen Zugang zum Haus des Opfers ermöglichen. Weiterhin werden eine große Anzahl von Smarthome Geräten als Massenware in Asien hergestellt und entsprechen oftmals nicht den Europäischen Qualitätsstandards in Bezug auf IT Sicherheit. Hier sind Hard- und Softwarekomponenten häufig von fragwürdiger Qualität oder aber sozusagen äb Werk"mit potentiellen Hintertüren versehen. Weitherhin können Angreifer durch Sicherheitslücken oder nicht vorhandene Schutzvorkehrungen beispielsweise Videomaterial aus dem Heim des Benutzers mitschneiden, Tonaufnahmen erhalten oder in extremen Fällen signalgesteuerte Türschlösser öffnen. Eine weitere Gefahr erwächst daraus, dass einige Geräte gesammelte Daten vor ihrer Verarbeitung zu einem Server im Herstellungsland senden und damit Regierungsstellen oder Firmen, die die Nutzerdaten des Benutzers verkaufen, Zugriff auf die so übertragenen Informationen erhalten [Inc18]. Diese Risiken und mögliche Gegenmaßnahmen sowie deren korrekte Umsetzung sind dem durchschnittlichen Anwender in der Regel kaum oder gar nicht bekannt.

1.2. Motivation

Ziel dieses Projekts ist es, von einem beliebigen Ort eine Verbindung in das eigene Heimnetz zu ermöglichen, welche nach dem letzten Stand der Technik verschlüsselt ist. Gewährt man dann den Zugriff auf diverse IoT Geräte danach nur noch aus dem eigenen Netzwerk, verhindert man so den unerlaubten Zugriff auf diese von außen. Zugleich schränkt man so die Gefahr einer „Man-in-the-Middle“ Attacke stark ein.

1.3. Umsetzung

Zur technischen Umsetzung verwenden wir verschiedene, quelloffene Bibliotheken – allen voran das aus der Schweiz stammende „strongSwan“. Mit diesem Kommandozeilen-Tool werden einige Funktionen wie der Schlüsselaustausch oder die Verbindungsinitialisierung deutlich vereinfacht. Der Nachteil von strongSwan liegt in auch der für technisch versierte Nutzer relativ hohen Einarbeitungszeit und Fehleranfälligkeit. Das Endergebnis des Projekts zeigt eine mögliche Konfigurationsvariante des „Roadwarrior“ Szenarios vor, bei dem sich ein dynamischer Klient mit einem statischen Server verbindet. Sie beginnt mit einer einfachen Implementierungsmöglichkeit über ein vor der Verbindung bei beiden Kommunikationspartnern eingetragenem Passwort und schließt mit der Verwendung von Zertifikaten ab.

Teil I.

Kryptografische Grundlagen

2. Pre-Shared Key (PSK)

Ein „Pre-Shared Key“ ist eine geheime Zeichenkette, die vor der Kommunikation zwischen zwei oder mehreren Teilnehmern über einen sicheren Kanal oder manuell ausgetauscht wird. Die Kommunikationspartner nutzen sie, um aus ihr einen Schlüssel zur Kodierung derjenigen Daten zu generieren, die sie miteinander austauschen wollen. Dieses Verfahren wird beispielsweise für die Anmeldung von Geräten in einem WLAN-Netzwerk angewandt (WPA2-PSK). Im Versuchsaufbau dieses Projekts nutzen wir es in Kombination mit dem „Extensible Authentification Protocol“, auch „EAP“. Die Paarung der beiden Algorithmen ist auch unter „EAP-PSK“ bekannt.

2.1. Aufbau

Je nach Anforderung der teilnehmenden Systeme ist der Aufbau des PSK unterschiedlich. Wählbare Parameter sind beispielsweise die Länge der zu verwendenden Zeichenkette sowie das Vorkommen von Sonderzeichen und/oder Zahlen im Schlüssel.

2.2. Generierung des PMK mit PBKDF2 und HMAC

Damit ein System verifizieren kann, dass sein Partner, mit dem es sich austauschen will, den gleichen PSK besitzt, müssen beide diese Information an den jeweils anderen übermitteln. Um zu gewährleisten, dass ein potenzieller Man-in-the-Middle Angreifer nicht in der Lage ist, diesen in Klartextform auszulesen, legen beide vorher Verschlüsselungsverfahren fest, um den PSK zu kodieren. Das Ergebnis dieser Verschlüsselung nennt man „Pairwise-Master-Key“.

2.2.1. PBKDF2

Das „Institute of Electrical and Electronics Engineers“ (IEEE) empfiehlt zur Generierung des PMK die von der „Internet Engineering Task Force“ (IETF) in RFC 2898 [Kal00] definierte „Schlüsselableitungsfunktion“ „PBKDF2“ (Password-Based Key Derivation Function 2).
Der PMK wird mit der Funktion

```
PMK = PBKDF2(PRF, P, S, c, dkLen)
```

und den Parametern

- PRF: Eine pseudozufällige Funktion
- P: Password
- S: Salt
- dkLen: Beabsichtigte Länge des abgeleiteten Schlüssels

erstellt.

WPA2 nutzt PBKDF2 beispielsweise mit den Parametern

```
PMK = PBKDF2(HMAC-SHA1, PSK, SSID des Netzwerkes, 4096, 256)
```

[IEE07].

2.2.2. HMAC

„HMAC“ (Hash-Based Message Authentification Code) ist ein Verfahren, um aus einem kryptografischen Schlüssel und einer Hash-Funktion einen Nachrichtenauthentifizierungscode zu berechnen [KCB97]. Dieser wird bei der Kommunikation zwischen zwei Partnern mit der Nachricht mitgeschickt. Der Empfänger dieser Nachricht hasht sie erneut mit dem spezifischen HMAC Verfahren des Absenders, und kann so die Integrität der Nachricht überprüfen [MVO96].
Ein Vorteil von HMAC ist, das die zu verwendende Hashfunktion beliebig ausgetauscht werden kann. Der resultierende MAC Algorithmus wird dann beispielsweise „HMAC-SHA1“ oder „HMAC-MD5“ genannt.

2.3. Limitierungen

2.3.1. Brute Force Angriff

Da ein PSK ein festes Passwort ist, ist er bei unzureichender Länge anfällig gegen eine „Brute-Force“-Attacke. Besitzt ein Angreifer genug Rechenleistung, so kann er alle möglichen Kombinationen von Buchstaben, Zahlen und Sonderzeichen generieren und kodieren, um sie anschließend mit dem PSK abgleichen.

2.3.2. Fundamentale Sicherheitsbedenken

Aufgrund der Tatsache, dass ein PSK allen teilnehmenden Kommunikationspartnern bekannt sein muss, verliert er seine Wirkung, wenn er durch einen Hackerangriff auf *mindestens einen* Verbindungsteilnehmer bekannt wird. Ist das der Fall, kann ein Hacker den gesamten Datenverkehr entschlüsseln.

3. EAP

„EAP“ (Extensible Authentification Protocol) ist in RFC 3748 [AB04] festgelegt und stellt ein Authentifizierungsprotokoll dar. Ein Authentifizierungsprotokoll beschreibt, wie Absender und Empänger von Daten ihre Identität gegenseitig legitimieren können. EAP unterstützt die Nutzung verschiedener Authentifizierungsmethoden. Zusätzlich existiert in RFC 5247 [AS08] eine weitere Standardvorgabe zur Nutzung und dem Transport von Schlüsselwerten, die die voherigen RFCs erweitert. Diese Schlüsselwerte werden von der verwendeten Methode erzeugt.
Da EAP nur Nachrichtenformate beschreibt, geben alle Protokolle, die es verwenden, an, wie EAP Nachrichten in ihrem jeweiligen Datensegment einzubetten sind. Dieses Verfahren wird Kapselung gennannt. Es ist in 3.4 beschrieben.
Obwohl EAP urpsrünglich für ein sicheres Einwählverfahren beim „Dial-up“ Internetzugriff gedacht war, nutzt man es ebenfalls in LAN- und Drahtlosnetzwerken.

3.1. Aufbau

Da EAP selbst nur einen Satz von Standardvorgaben zur universellen Authentifizierung bereitstellt, enthält es sogenannte „EAP-Methoden“. Während EAP selbst sich also um den Transport und die Verarbeitung von Daten und Parametern der EAP-Methoden kümmert, implementieren diese ihre jeweiligen Authentifizierungsmethoden. Das wiederum ermöglicht das flexible Austauschen derselben. Zudem ermöglicht EAP über einen sogenannten „EAP-Server“ den Einsatz von Authentifizierungsvarianten, die dem Authentifikator nicht bekannt sind. Das bedeutet, für die Verbindung von EAP-Server zu Klient eine andere Verschlüsselungsmethodik zu nutzen als von Klient zu Server.

3.2. EAP-Methoden

Zu den gängisten EAP Methoden zählen EAP-MD5, EAP-SIM, und EAP-TLS. Die jeweiligen Methoden sind in RFCs der „Internet Engineering Task Force“ (IETF) definiert. Außerdem gibt es einige herstellerspezifische EAP Methoden, welche durch den Hersteller selbst definiert und eingesetzt werden.

3.3. EAP-IKEv2

EAP-IKEv2 basiert auf dem „Internet Key Exchange Protokoll Version 2“, welches in 4.4 erläutert wird. EAP-IKEv2 ist von Tschofenig und Kroeselberg [TK08] in RFC 5106 spezifiziert. Dieses erläutert wechselseitige Authentifizierungsmethoden zwischen Klient und Server und liefert eine Methode zum Verwalten von Verbindungssitzungen. Zur Authentisierung können neben PSKs auch Zertifikate verwendet werden. Zudem unterstützt EAP-IKEv2 unter anderem ein Verfahren, nach einem unbeabsichtigten Verbindungsabbruch diese schnell wiederherzustellen.

3.4. Kapselung

EAP agiert direkt auf der Sicherungsschicht des OSI-Modells [Kle19]. Das heißt, dass es keine Kenntnisse über Protokolle benötigt, die IP-basiert sind. Stattdessen „umschließen“ Protokolle, welche sich auf höheren Schichten befinden, die EAP Pakete mitsamt ihrer Kopfzeile. Die Daten in der Körperzeile dieses Protokolls, welche zum Beispiel von Router an Server verschickt werden, sind dann mit der angegebenen EAP-Methode verschlüsselt.

4. Internet Protokol Security (IPsec)

„IPsec“ ist ein Satz von Protokollen, mit dessen Hilfe eine gesicherte Übertragung von Daten in nicht gesicherten IP-Netzwerken realisiert wird. IPsec kümmert sowohl um die Authentifizierung zwischen mehreren Kommunikationsteilnehmern als auch um die Verschlüsselung der Daten, die sie senden. Informationen, Spezifikation und Dokumentation von IPsec können in RFC 2401 [KA98a] sowie in der aktualisierten Version, RFC 4301 [KS05], gefunden werden.

4.1. Aufbau

Um eine sichere Verbindung zu gewährleisten, nutzt IPsec die Protokolle „Authentification Header“ und „Encapsulating Security Payload“ in Kombination.

4.1.1. Authentification Header (AH)

Der Authentification Header ist ein verbindungsloses Protokoll, das durch einen generierten Hash-Wert und ein „Shared Secret“ [KA98b] die Integrität von IP Datagrammen gewährleistet. Optional ist es in der Lage, für einzelne IP-Datagramme in einem Datenstrom Sequenznummern zu verwalten, um vor sogenannten „Replay“-Angriffen zu schützen. Bei einem solchen Angriff zeichnet ein Hacker einen bereits gesendeten Datenfluss auf und verschickt ihn erneut an den ursprünglichen Empfänger, um sich beispielsweise mit einem Hashcode fälschlicherweise zu identifizieren.
Das Authentification Header Protokoll ist nur in der Lage, statische Bestandteile eines IP-Paketes zu schützen. Wird ein IP-Paket jedoch von Sender zu Empfänger über einen Router versendet, auf dem die „Netzwerkadressübersetzung“ aktiviert ist, verändern sich die eigentlich statischen Daten des Paketes. Die Integrität kann nicht mehr gewährleistet werden und eine Authentifizierung ist somit nicht mehr möglich. Um dieses Problem zu lösen, nutzt man eine Kombination aus Authentiction Header und ESP [KA98b] [Hut+05].

4.1.2. Encapsulating Security Payload (ESP)

ESP stellt die Sicherung aller Eigenschaften des „CIA“-Modells [And11] bereit: Vertraulichkeit, Authentifizierung des Datenursprungs und, wie der „Authentification Header“, die Integrität von versendeten Datenpaketen. Im Gegensatz zum AH ist es bei ESP jedoch möglich, optionale Felder in der IP-Datagramm Kopfzeile bei der Verwendung auszulassen. Damit kann ESP auch bei einer Datenweiterleitung über NAT verwendet werden [KA98c].

4.1.3. Security Association (SA)

Eine „Security Association“ ist ein Teil des IPsec-Protokolls, welches den Austausch von verwendeten Algorithmen und öffentlichen Schlüsseln, beziehungsweise das zur Generierung des Schlüssels benötigte Schlüsselmaterial, handhabt und verwaltet [Mas02]. Neben den Parametern einer Gültigkeitsdauer der Verbindung werden auch ein Schlüssel zur Identifizierung der Sitzung

oder Teile von Publik-Key-Infrastrukturen übertragen. Das Rahmenwerk, welches IPsec für diese Aufgabe verwendet, heißt „ISAKMP“ (Internet Security Association and Key Management Protocol) [Mau+98]. Es definiert allerdings nur, wie Authentifizierung und der Schlüsselaustausch zu nutzen sind, legt aber keine spezifischen Algorithmen fest (vgl: [HC98]. Es wird deswegen auch als „unabhängig vom Schlüsselaustauschalgorithmus“ bezeichnet.

4.2. IKE

„IKE“ (Internet Key Exchange) ist ein Protokoll, welches beschreibt, wie zwei Kommunikationspartner Verschlüsselungsmaterial aushandeln. Es ist definiert in RFC 2409 der IETF [HC98], nach dem sich dieser Abschnitt richtet. IKE ist ein zusammengesetztes Protokoll aus den Protokollen ISAKMP, Oakley und SKEME. Die genaueren Funktionsweisen dieser Protokolle sind in Maughan u. a. [Mau+98], Orman [Orm98] als auch in Krawczyk [Kra96] zu finden. IKE ermöglicht die Erzeugung einer Security Association (siehe 4.1.3. Es ist in zwei Versionen vorhanden: IKE(v1) und IKEv2.
Der Verbindungsaufbau findet in zwei Phasen statt:
In Phase 1 authentifizieren sich die Verbindungspartner gegenseitig. Danach bauen sie über das sogenannte „ISAKMP Security Association (SA)“ Protokoll einen sicheren Weg auf, um SA Daten auszutauschen und einen Schlüssel zu verhandeln. Phase 2 beschreibt den Vorgang, in dem verschiedene Dienste der Kommunikationspartner den in Phase 1 aufgebauten, sicheren Tunnel nutzen. In der Praixs wäre das beispielsweise der IPsec-Dienst.

4.3. Probleme bei der Anwendung von IPsec und IKEv1

Enige Sicherheitsforscher erachten IPsec aufgrund seiner Komplexität als zu unsicher. Ferguson und Schneier [FS00] gehen sogar soweit, zu behaupten, dass „kein IPsec System [aufgrund der Implementierung zu vieler Konfigurationskonstellationen und Optionen] sein Ziel erreicht, ein hohes Level an Sicherheit zu bieten“. Sie kritisieren weiterhin die Tatsache, dass IPsec „zu unterschiedliche viele Wege bietet, um das gleiche Ziel zu erreichen“, und bemängeln die Qualität der vorhandenen Dokumentation.

In ihrem Paper bemängeln Ferguson und Schneier [FS00] auch IKE. Sie stellen zwei verschiedene Angriffsszenarien dar, für die IKEv1 anfällig ist. Es handelt sich dabei um die Angriffsmöglichkeiten „Reflection Attack“ und „Proposal Attack“.

4.3.1. Reflection Attack

IKE ist laut Ferguson und Schneier [FS00] deswegen anfällig, weil der Weg zur Generierung der Hash Werte, die es zur Authentisierung verwendet, auf Klient- und Serverseite gleich sind. In diesem sogenannten „Challenge-Response Authentifizierungssystem“ müssen beide Angreifer das Shared-Secret kennen, um sich zu authentifizieren.
Die Reflection-Attack beschreibt die Möglichkeit, eines Angreifers, sich zu authentisieren, ohne das Shared-Secret zu kennen. Er versucht, eine Verbindung Con_1 aufzubauen und sendet eine sogenannte „Challenge“ C_1, beispielsweise einen String. Daraufhin antwortet der Server mit einer eigenen Challenge C_2, die mit dem Shared-Secret S verschlüsselt ist. Der Angreifer baut nun eine zweite Verbindung Con_2 auf und sendet als Antwort in Con_2 anstatt einer eigenen Challenge C_2, die er in Con_1 erhalten hat. Der Server verschlüsselt diese wiederum mit dem Shared-Secret S.

An dieser Stelle liegt die Essenz des Angriffs: C_2 wurde nun zwei mal mit S verschlüsselt. Sendet der Angreifer C_2 nun in Con_1 an den Server zurück, denkt der Server, der Angreifer würde das Shared-Secret kennen. Der Angreifer ist damit erfolgreich authentifiziert (vgl [CAP18]).

4.3.2. Proposal Attack

Sendet ein Klient eine Verbindungsanfrage mit einer Liste von Verschlüsselungsalgorithmen, die im Schlüsselaustausch der SA verwendet werden sollen, so kann ein Angreifer, der die Antwort des Servers auf diesen SA-Request abfängt, laut Ferguson und Schneier [FS00] die schwächste unter diesen auswählen und die originale Nachricht weitersenden. Dieses Verfahren funktioniert, weil die Berechnung des Hash-Wert, der zur Integritätsprüfung genutzt wird, die SA-Reply des Beantworters nicht enthält. Der Klient verwendet die vom Hacker manipulierte SA-Reply, und verwendet nun den vom Angreifer ausgewählten Verschlüsselungsalgorithmus. Ist dieser schwach genug, könnte der Angreifer den verwendeten Schlüssel durch eine Brute-Force Attacke wiederherstellen.

4.4. IKEv2 und Verbesserungen gegenüber IKEv1

Die zweite Version von IKE ist in RFC 4306 [Kau98] spezifiziert. Dieser RFC markiert die RFC Threads 2407 [Pip98], 2408 [Mau+98] und 2409 [HC98], welche sich auf IKEv1 und ISAKMP beziehen, als veraltet. RFC 4306 vereint den Inhalt dieser RFCs und ihre eventuellen Neuerungen in sich selbst.
Im Gegensatz zu IKE(v1) bietet IKEv2 die Möglichkeit, X.509 Zertifikate einzubinden. Diese sind in 5.1 näher erläutert. Weiterhin erlaubt die Spezifikation des Protokolls auch die Möglichkeit, die Sicherheit bei einer Verbindung über einen NAT aufrecht zu erhalten, indem es die IKE und ESP Pakete in ein UDP Paket verpackt und es über den NAT relevanten Port 4500 versendet. Dieses Verfahren nennt man „NAT Traversal“ [AD04].

ÏKEv2 vereinfacht zudem die IPsec Interoperabilität zwischen verschiedenen VPN Anbieternünd ist schneller, wenn es darum geht den verwendeten Schlüssel zu erneuern [CIS13].

Nach seiner Veröffentlichung modernisierte das IETF IKEv2, in dem es einige Erweiterungen nachrüstete. Zu diesen zählen beispielsweise „IKE session resumption“ [ST10], ein Verfahren zur Sitzungswiederherstellung bei Verbindungsabbruch und die Möglichkeit, auf Zertifikate zu verzichten und IKEv2 nur mit EAP basierten Passwörtern zu betreiben [ETS10].

5. Zertifikate

Zertfikate werden zur Authentifizierung von Rechnern und Personen genutzt und bilden einen Grundpfeiler von Public-Key-Infrastrukturen [Kuh+01]. Ein Zertifikat ermöglicht es einem Rechner, seine Identität gegenüber einem Partner zu beweisen, indem beide einen vorher festgelegten Parameter, in diesem Sinne dem Zertifikatshersteller und -signierer, vertrauen.
In den folgenden Abschnitten wird ein grober Überblick über den Aufbau von Zertifikaten gegeben.

5.1. Aufbau

Der von der „International Telecommunication Unit“ (ITU) veröffentlichte Standart X.509 [Hou+02] [Coo+08] gehört zu den meistgenutzten Arten von Zertifikaten. Er befindet sich aktuell in der dritten Version. In Abbildung 5.1 ist die Struktur eines X509-Zertifikates zu sehen. Hier erkennt man auch die Erweiterungen, die die Version drei mit sich brachte.

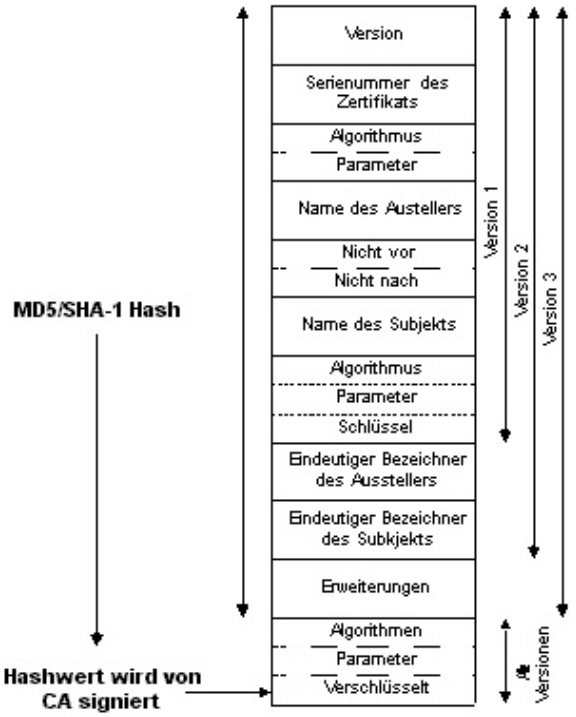

Abbildung 5.1.: Aufbau eines X509 Zertifikates
Blomenkamp u. a. [Blo+02]

Im folgenden befindet sich eine Liste mit Erläuterungen für die wichtigsten Felder. Eine vollständige Ausführung findet sich in „Kryptographie und sichere Kommunikation in heterogenen Netzwerken: IPSec“ [Blo+02].

Version: Die Version dieses Zertifikates. Mit Hilfe dieser Information ist man in der Lage, Felder, die erst in neueren Versionen hinzugefügt wurden, zu erkennen.

Seriennummer: Die Seriennummer eines Zertifikates dient dazu dieses innerhalb der der aktuellen Infrastruktur eindeutig zu identifizierern.

Algorithmus: Gibt den Algorithmus an, mit dem der Zertifikatsaussteller dieses Zertifikat signiert hat.

Schlüssel: Hier wird der öffentliche Schlüssel des Zertifikatinhabers, also der Nutzers/Rechners notiert

Signatur: Am Ende folgt die Signatur der „Certificate Authority“ (CA)

5.2. Einsatzzwecke

Wie bereits erwähnt werden Zertifikate zur Authentifizierung verwendet. Im Alltag begegnet man ihnen am häufigsten in Browserprogrammen. Hier werden Zertifikate genutzt, um die Authentizität einer Webseite gegenüber dem Internetnutzer sicherzustellen.
Diese Zertifikate werden von namhaften Unternehmen wie „digicert“ oder „GlobalSign“ verteilt. Diese wiederum sind im Besitz von Verträgen mit namhaften Browser-Herstellern, wie Google oder der Mozilla-Foundation, um ihre Grundzertifikate in eben jene Browser von Haus aus integrieren zu lassen.
Innerhalb dieser Projektarbeit werden die Zertifikate von strongSwan verwendet, was somit einen sicheren Verbindungsaufbau ermöglicht.

Teil II.

Sichere Public-Key-Infrastruktur

6. Motivation

Die Verwendung einer sicheren Public-Key-Infratstruktur ist für die Nachrichtenübertragung heutzutage unerlässlich. Neben klaren Vorteilen, wie der abhörsicheren Übertragung von Informationen und der kryptografisch sicheren Authentisierung von Nutzern, bietet die technische Implementierung einen deutlichen Komfortzuwachs. So können Verschlüsselungs- und Authentifizierungsprozesse mit wenig Aufwand implementiert und eingesetzt werden. In diesem Kapitel wird die technische Umsetzung einer solchen Infrastruktur mit Hilfe des RaspberryPi vorgeführt.

7. Installation des RaspberryPi

7.1. Hardware

Als Hardwaregrundlage dient hier ein RaspberryPi 3. Dieser ist optimalerweise per Gigabit-LAN an den entsprechenden Router anzuschließen. Im unten aufgeführten Fallbeispiel handelt es sich um eine FritzBox 7350 mit der zu diesem Zeitpunkt aktuellen Software-Version 7.0.

7.2. Software

7.2.1. Raspbian

Für den RaspberryPi sind eine Vielzahl an Betriebssystemen auf Linux-Basis verfügbar. Durch den hohen Grad an Kompatibilität und der verfügbaren Dokumentation wurde im Fallbeispiel „Raspbian“ als Betriebssystem verwendet. Jenes basiert auf der Linux Distribution Debian [GNU19] in der Version „Stretch“ , welches für den Einsatz auf dem RaspberryPi optimiert wurde und über zusätzliche, vorinstallierte Softwarepakete, wie Python, Java oder Mathematika verfügt. Um die Auslastung des Raspberry möglichst gering zu halten, wurde speziell die Unterversion „Raspbian Lite“ verwendet, welche auf einige, im Fallbeispiel nicht genutzte Softwarepakete und eine grafische Oberfläche verzeichtet und somit Rechenleistung spart.
Die vollständige Anleitung zur Installation von Raspbian kann unter `https://www.raspberrypi.org/documentation/installation/installing-images/README.md` gefunden werden.

7.2.2. SSH Zugriff

Um im konkreten Anwendungsbeispiel über die Kommandozeile auf den RaspberryPiI zugreifen zu können, wird ein Netzwerkprotokoll, das sich „SSH“ (Secure Shell) nennt, verwendet. Dieses benutzt ein sogenanntes „Public-Key-Verschlüsselungsverfahren“, um eine sichere Verbindung zwischen zwei Klienten herzustellen [YL06]. Auf Debian Systemen, und somit auf Raspian Lite, wird aus Lizenzgründen die Implementierung „OpenSSH“ (OpenBSD Secure Shell) verwendet [Ope19].
Um es dem RaspberryPi zu ermöglichen, passiv auf eventuelle SSH Zugriffe zu warten, wird das OpenSSH Programm als sogenannter „Service“ (Dienst) durch den auf Raspbian Lite vorinstallierten „Daemon “ „systemd“ aktiviert. Dieser enthält den „System and Service Manager“, ein Programm, das sich um die automatische Ausführung von registrierten Diensten beim Systemstart und die Protokollierung von möglichen Fehlern kümmert.

Die tatsächliche Aktivierung des SSH-Dienstes lässt sich mithilfe des Befehls `raspi-config` bewerkstelligen, welcher ein interaktives Kommandozeilenprogramm zur Konfiguration des Raspberry bietet. Aus Sicherheitsgründen ändern wir den SSH-Port des RaspberryPi von 22 auf den willkürlich gewählten Port 3000. So erschwert man beispielsweise Angreifern, die die Standardports einer IP überprüfen, das Sammeln von Informationen über das Gerät.
In Raspbian Lite wird dies durch eine Änderung an der folgenden Datei ermöglicht:

```
$ sudo nano /etc/ssh/sshd_config
```

Hier ändern wir nun die Zeile mit der Variable „Port“. Daraufhin muss der SSH-Service des Raspberry neugestartet werden:

```
$ sudo systemctl restart ssh
```

8. Einrichtung eines Dynamic Name Server

Ein „DNS“ (Dynamic Name Server) übersetzt Domainnamen wie „www.google.de“ in die entsprechende IP-Adresse. Damit auf den RaspberryPi extern zugegriffen werden kann, wird eine konkrete IP-Adresse benötigt. Durch die willkürliche Änderung dieser, zum Beispiel bei Neuanmeldung des Routers beim Internetanbieter, entsteht jedoch die Problematik, Konfigurationselemente des RaspberryPi oder des zugreifenden Klienten manuell anpassen zu müssen. Diese lösen wir durch die Verwendung eines „DynDNS“ Dienstes. Der ausgewählte Dienst, hier „No-IP“, wird an der eingesetzten FritzBox registriert, welche die IP des RaspberryPi an den Dienst überliefert und diese bei Änderung aktualisiert. Der RaspberryPi besitzt eine statische MAC-Adresse, über die er von der FritzBox identifiziert werden kann. Der Vorteil der oben aufgeführten Integrierung besteht in der Tatsache, dass die durch No-IP verfügbare, globale URL statisch ist, und bei einem Zugriff von außen die angegebene URL auf die aktuelle IP-Adresse des RaspberryPi aufgelöst wird, um so einen dauerhaften Eintrag in verschiedene Konfigurationsdateien zu ermöglichen.

8.1. Einrichtung von No-IP

No-IP bietet eine Web-Oberfläche, um einen Hostnamen in Verbindung mit einer Domain zu erstellen.

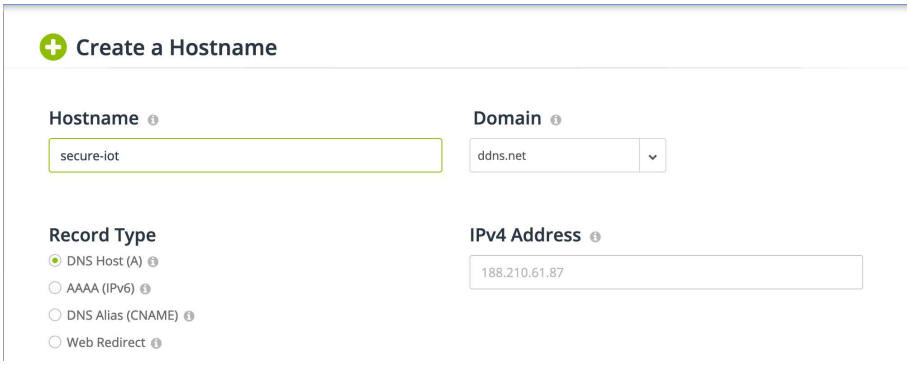

Abbildung 8.1.: Registrierung eine Hostnamens bei No-IP

Das Feld „IPv4 Adress“ ist leer zu lassen, da unser Zielgerät nicht über eine statische IPv4 Adresse verfügt.
No-IP bietet die Möglichkeit, der Installation der „Dynamic Update Client“ Software, welche den DynDNS Server in regelmäßigen Zeitabständen über die IPv4 Adresse des Gerätes informiert, auf dem sie installiert ist.

8.2. Einrichtung von DynDNS in der FritzBox

Aufgrund einfacherer Konfiguration von Port-Weiterleitungen und dem Vermeiden von Schwierigkeiten, die die Firewall des Raspberry Pi verursachen könnte, haben wir uns jedoch für die

direkte Konfiguration des DynDNS Services in der Fritz!Box entschieden. Dieses Vorgehen ist flexibler, weil der DynDNS Service dann vollkommen unabhängig vom Zielgerät kalibriert werden kann. Bei dem Gebrauch von mehreren Geräten im gleichen Netzwerk, die unterschiedliche DynDNS Adressen nutzen sollen, ist dieses Vorgehen allerdings nicht möglich.

Die Einrichtung des DynDNS Services in der Fritz!Box ist relativ einfach gestaltet. In der zum Zeitpunkt der Veröffentlichung aktuellen Version „07.12“ findet man das Konfigurationselement im Menü „Internet“ unter dem Reiter „Freigaben“.

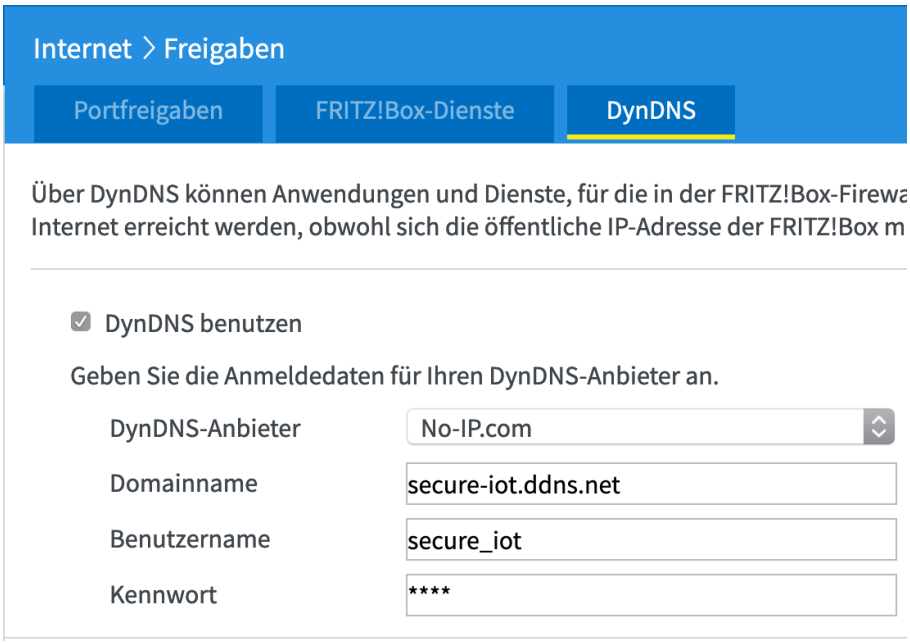

Abbildung 8.2.: Konfiguration des No-IP DynDNS Servers in der Fritz!Box

8.3. Portweiterleitungen für SSH in der Fritz!Box

Um von außen, also über die DynDNS Domain, auf den RaspberryPi zugreifen zu können, muss man den Port am Router in Verbindung mit dem Zielgerät freischalten, den SSH benutzt. Aufgrund der Änderung des SSH-Ports des RaspberryPi muss also ein beliebiger Port in der Fritz!Box freigegeben werden, der den gesamten Datenfluss auf den von uns in 7.2.2 gewählten Port 3000 umleitet. Dieser Port muss dann bei einem SSH-Zugriff auf den RaspberryPi ebenfalls explizit angegeben werden, da SSH ansonsten den Standard Port 22 verwendet.
Befindet man sich im Menü „Internet“ und dem Untermenü „Freigaben“, muss man dort den Reiter „Port-Freigaben“ auswählen. Die entsprechenden Freigaben am RaspberryPi richtet man mit einem Klick auf den Knopf „Gerät für Freigaben hinzufügen“ ein.

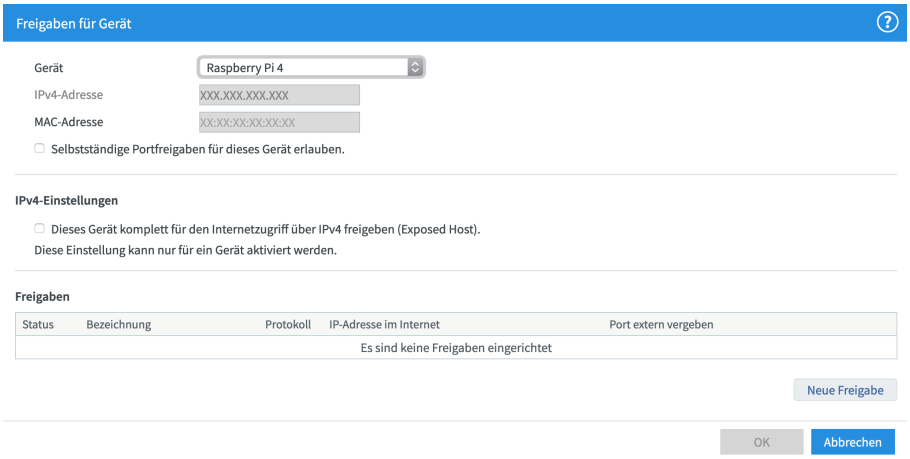

Abbildung 8.3.: Konfiguration des No-IP DynDNS Servers in der Fritz!Box

Unter „Neue Freigabe“ erstellen wir nun eine Freigabe des einzelnen Ports 3000. Unter „Port an Gerät“ trägt man den Port ein, den die Anwendung, in diesem Fall SSH am RaspberryPi entgegen nehmen soll. Das Feld „Port extern gewünscht (IPv4)“ hingegen füllt man die Port-Nummer des Ports ein, den die Fritz!Box nach außen hin freigeben soll. Da SSH ein TCP Dienst ist [YL06] müssen wir das entsprechende Protokoll ebenfalls auswählen. Wichtig ist ebenfalls, dass der Haken bei „Freigabe aktivieren“ gesetzt ist.

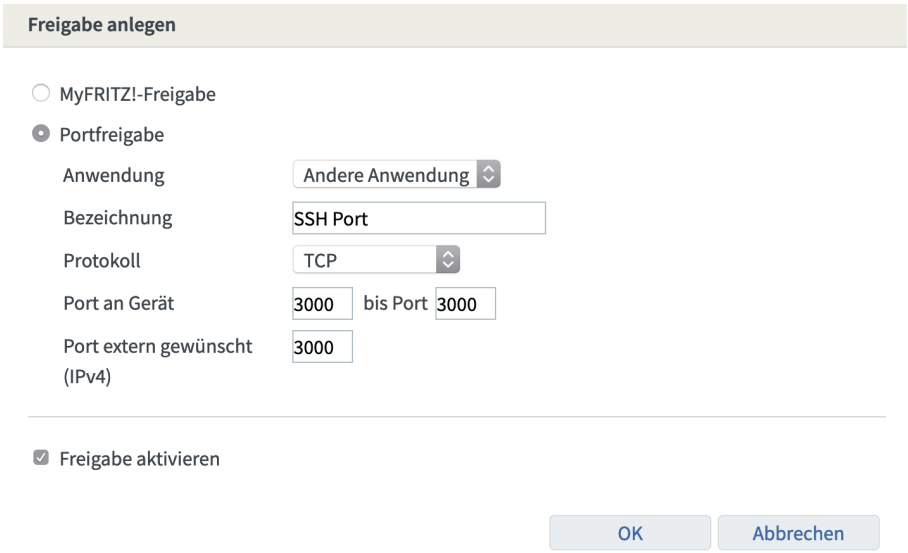

Abbildung 8.4.: Konfiguration des No-IP DynDNS Servers in der Fritz!Box

Nach dem Drücken auf „OK“ und der darauffolgenden Weiterleitung auf die „Port-Freigaben“ Seite erkennt man an den grünen Schaltflächen, ob die Ports freigegeben sind.

9. strongSwan

„strongSwan is an OpenSource IPsec implementation.“ [Str18] Das im Jahre 2005 gestartete und hauptsächlich von der „Hochschule für Technik Rapperswil“ entwickelte Projekt bietet die Möglichkeit Verbindungen per IKEv1/IKEv2 auf verschiedenen Geräten und Plattformen aufzubauen.

9.1. Installation

Die Installation von strongSwan auf unserem RaspberryPi kann mit dem mitgelieferten Paketmanager „apt“ vollzogen werden:

```
$ sudo apt-get install strongswan libcharon-extra-plugins
```

9.2. Konfiguration

In diesem Abschnitt wird die Konfiguration von strongSwan für den von uns verwendeten Anwendungsfall beschrieben. Dieser basiert auf dem von strongSwan „Roadwarrior“ getauften Szenario. Detailliertere Informationen zur Konfiguration von Verbindungen und Anpassung der Parameter dieser finden sich unter `https://wiki.strongswan.org/projects/strongswan/wiki/ConfigurationFiles`.

9.2.1. Übersicht

Die Konfiguration von strongSwan teilt sich grundlegend in verschiedene Dateien auf, welche unterschiedliche Aspekte übernehmen. Damit wird eine gewisse Modularität des Systems ermöglicht. Außerdem können somit Fehler auf einzelne Dateien eingegrenzt werden.
In diesem Abschnitt stellen wir kurz die einzelnen Dateien sowie ihre Funktionen vor, bevor wir auf die spezifischen Konfigurationen eingehen.

Dateien

„ipsec.secrets“
Hier finden sich Informationen zur Authentifizierung mit den möglichen Verschlüsselungsverfahren. Probleme mit einem falschen Nutzernamen, Passwort oder einer fehlenden Schlüsseldatei lassen sich somit an dieser Stelle beheben.

„ipsec.conf“
Die Verbindungsmöglichkeiten mit ihren Parametern werden hier festgelegt. Die Abstimmung zwischen einem entfernten Rechner und dem eigenen über die zu verwendenden Parameter erfolgt über diese Konfigurationen.

„sysctl.conf“
Diese Datei ermöglicht die Anpassung des Linux-Kernels am laufenden Rechner. Einstellungen wie die Paketweiterleitung werden hier vorgenommen.

Ordner

„/etc/ipsec.d/certs/“
Die Nutzerzertifikate werden hier aufbewahrt.

„/etc/ipsec.d/cacerts/“
Das Zertfikat der „Certificate Authority“(), welche das Zertifikat des Verbindungspartners signiert hat, liegt hier.

„/etc/ipsec.d/private/“
Die RSA-Schlüssel zu den einzelnen Zertifikaten liegen in diesem Ordner. Dieser ist nur mit root-Rechten betret- und änderbar.

9.2.2. ipsec.secrets

Diese Systemweite Variante der Datei findet sich auf den Betriebssystemen und auch auf Raspian unter `/etc/` . Die nutzerspezifische Version findet man unter `/usr/local/etc/` . Sie enthält die Konfiguration der Sicherheitsmerkmale für die verwendeten Authentifizierungsverfahren. Sie sollte sich zwischen der Server-Seite und der Client-Seite in dem Sinne unterscheiden, dass der Client nur die Merkmale des Servers kennt, der Server jedoch die aller möglichen Nutzer.
In unserem hier implementierten Beispiel ist die Datei größtenteils identisch, da wir eine 1:1-Verbindung erzeugen – ein Rechner verbindet sich mit unserem Server, dem RaspberryPi. Daher wird hier in diesem Abschnitt die ipsec.secrets Datei nur einmal behandelt, sie sollte jedoch auf beiden Seiten vorliegen und entsprechend konfiguriert werden, beispielsweise beim Einfügen weiterer Nutzer.
Weitere Informationen zur Konfiguration und den Möglichkeiten dieser Datei finden sich unter `https://wiki.strongswan.org/projects/strongswan/wiki/IpsecSecrets`

9.2.3. ipsec.conf

Auch diese Konfigurationsdatei ist unter `/etc/` zu finden.
Anders als bei der ipsec.secrets finden sich hier auch bei unserem einfachen Verbindungsszenario grundlegende Unterschiede in der Konfiguration, weswegen hier beide Seiten differenziert angesprochen werden.

Grundlegender Aufbau der Datei

Es ist zu beachten, dass in folgendem Kontext mit „left“ immer die aktuelle Seite der Verbindung und mit „right“ der entsprechende Kommunikationspartner gemeint ist. Merken lässt sich dies mit „l“eft als „l“ocal und „r“ight als „r“emote.

Die einzelnen Verbindungen – sog. „Connections“, kurz als „conn“ geschrieben, besitzen Namen, die eine Identifikation der Kommunikation erlauben. Bis auf Unterscheidung von Verbindungen und der Hilfe bei der Fehlersuche haben diese aber keine relevante Bedeutung für den technischen Verbindungsablauf.
Verbindungen lassen sich, ähnlich der objektorientierten Programmierung, „erweitern“ und ihre Parameter „überschreiben“. Somit ist es beispielsweise möglich, eine Grundverbindung zu erstellen, welche alle notwendigen Grundparameter an einer Stelle definiert, diese entsprechend zu erweitern und Merkmale, die sich unterscheiden, zu ändern. Damit ist sichergestellt, dass auch beim Anlegen von vielen verschiedenen Verbindungen für diverse Einsatzzwecke die wichtigsten Konfigurationsparameter an einer zentralen Stelle verwaltet werden können.

10. Naiver Verbindungsansatz mit Pre-shared Key

Die folgenden beiden Abschnitte zeigen die Konfigurationsdateien auf der Server- und der Client-Seite und sprechen die wichtigsten Verbindungen und ihre Parameter an. Details zu einzelnen Parametern sind weiterhin in der strongSwan-Dokumentation nachzulesen.

10.1. Inhalt der ipsec.secrets

```
1 include /var/lib/strongswan/ipsec.secrets.inc
2
3     : PSK "our-pre-shared-key"
```

Zeile 3 definiert das zu nutzende Passwort für eine Authentifizierung mit dem „Pre-Shared-Key“-Verfahren. Dieses Kennwort muss, wie der Name es bereits andeutet, vor dem Aufbau der Verbindung zwischen den Verbindungspartnern ausgetauscht werden. Wie bereits in Kapitel 2 erwähnt, gilt dieses Verfahren aufgrund der Gefahr einer „MITM“ Attacke beim initialen Passwortvergleich zwischen Server und Klient als unsicher.

Das Fehlen eines Nutzernamens in dieser Zeile erlaubt es uns als Server das Kennwort jedes anfragenden Clients zu überprüfen und eine Verbindung zu erlauben – unabhängig von dem Nutzernamen dieses Clients.

10.2. Inhalt der ipsec.conf

10.2.1. Server-Seite

```
1  # ipsec.conf - strongSwan IPsec configuration file
2
3
4  config setup
5      uniqueids = no
6
7  conn %default
8      mobike=yes
9      dpdaction=clear
10     dpddelay=35s
11     dpdtimeout=200s
12     fragmentation=yes
13
```

```
14 conn htw
15     auto=add
16     keyexchange=ikev2
17     eap_identity=%any # accepts every user with right password
18     left=%htwsaariot.ddns.net # identifies as htwsaariot.ddns.net
19     leftsubnet=%any
20     leftid=@htwsaariot.ddns.net
21     leftauth=psk # authentifies via pre-shared key (see ipsec.
           secrets)
22     right=%any
23     rightsourceip=%dhcp
24     rightauth=eap-mschapv2 # remote host authentifies via eap
25     rightdns=8.8.8.8,8.8.4.4 # dns-servers for the remote host
26     rightid=%any
27     leftfirewall=yes
28
29 conn htw-enc
30     also="htw"
31     ike=aes256-sha2_256-modp1024!
32     esp=aes256-sha2_256!
33
34 include /var/lib/strongswan/ipsec.conf.inc
```

„Grundverbindung ‚htw‘“

Zeile 14 Diese Verbindung ist unsere Grundverbindung und stellt alle wichtigen Parameter für den Verbindungsaufbau bereit.

Zeile 16 Diese Einstellung spezifiert das IKEv2-Verfahren 3.3 für den Schlüsselaustausch

Zeile 17 Hiermit wird die Validierung der Nutzeridentität bei einem Verbindungsaufbau per EAP unterdrückt, d.h. jede Identität mit den richtigen Zugangsdaten (Password, Zertifikat, ...) hat Zugriff.

Zeile 18 Diese Zeile gibt uns, genau wie Zeile 20, die Identität unseres Servers an.

Zeile 19 Mit dem Left-Subnet ist das Subnetz unserer Server-Seite gemeint und welches für die Vergabe erlaubt ist. Mit der hier eingetstellten Konfiguration sind alle Subnetze zugelassen und der gesamte Datenverkehr wird durch den Raspberry weitergeleitet.

Zeile 21 In dieser Verbindung authentifiziert sich diese Seite mit einem Pre-Shared-Key.
„Anmerkung: Die andere Seite muss in ihrer Konfiguration entsprechend eine Verbindung haben, die es der rechten Seite erlaubt sich mit Pre-Shared-Key zu authentifizieren.“

Zeile 22 Alle Kommunikationspartner werden akzeptiert, es ist keine spezielle Identität notwendig. (siehe ebenfalls Zeile 28)

Zeile 23 Vergibt die IP-Adresse des Kommunikationspartners mit Hilfe des DHCP-Servers. Stattdessen kann an dieser Stelle auch eine Netzwerk-Adresse und ihre Netzwerkmaske angegeben werden, aus der die Adressen gewählt werden sollen.

Zeile 24 Unser Kommunikationspartner soll sich mit EAP authentifizieren.

Zeile 25 Die DNS-Server welche im Tunnel durch die Klienten verwendet werden sollen. Wir verwenden aus Testzwecken den Google DNS Server. Hier könnten auch eigene Server definiert werden um ein weiteres Sicherheitslevel einzurichten, in dem die Einträge in diesen Servern streng überprüft werden.

Zeile 27 Hiermit kann im Prinzip angegeben werden, ob unsere linke Seite, also in dem Falle der RaspberryPi, eine Art IP-Masquerading nutzt. Dies können zum Beispiel Regeln in den IP-Tables sein, welche bestimmte Pakete verteilen. Auf den meisten Servern ist hiermit zu rechnen, daher wird die Nutzung der Option empfohlen.

„Erweiterung um Verschlüsselung mit ‚htw-enc'"

Zeile 30 Diese Zeile ist essentiell bei der „Vererbung" beziehungsweise. Erweiterung von Verbindung. Sie kann wie eine „Präprozessor-Direktive" in der Programmiersprache C verstanden werden und kopiert alle Einstellungen der angegebenen Verbindung an diese Stelle.

Zeile 31 Es wird angegeben, welche Verschlüsselungen für IKE 3.3 verwendet werden sollen. Die Angabe von mehreren Verfahren erlaubt es den beiden Kommunikationspartnern sich auf ein für beide verfügbares Verfahren zu einigen. Dabei sind die Verfahren per „-" getrennt.

Zeile 32 Ähnlich wie Zeile 33. Gibt die Verschlüsselungsverfahren an.

„Anmerkung zu den Zeilen 31 und 33: Das Ausrufezeichen am Ende der Liste der Algorithmen gibt an, dass nur diese Verfahren gültig für den Verbindungsaufbau sind. Dieses Ausrufezeichen wird auch entsprechend ‚Strict Flag' genannt."

10.2.2. Client-Seite

```
1  # ipsec.conf - strongSwan IPsec configuration file
2
3
4  conn %default
5      mobike=yes
6      dpdaction=clear
7      dpddelay=35s
8      dpdtimeout=200s
9      fragmentation=yes
10
11 conn htw
12     right=htwsaariot.ddns.net
13     rightid=%any
14     rightsubnet=0.0.0.0/0
15     rightauth=psk
16     leftsourceip=%config
17     eap_identity=pi
18     auto=add
19
20 conn htw-enc
```

```
21      also="htw"
22      ike=aes256-sha2_256-modp1024!
23      esp=aes256-sha2_256!
24      leftauth2=eap-mschapv2
25
26
27  include /var/lib/strongswan/ipsec.conf.inc
```

In der folgenden Aufzählungen werden nur Zeilen erläutert, die sich sich essentiell unterscheiden oder deren gesonderte Erläuterung einen Mehrwert zum Verständnis beiträgt. Die meisten Angaben entsprechend der Server-seitigen Konfiguration und unterscheiden sich nur in der Angabe der Parameterseite (left vs. right), da strongSwan so passende Verbindungen wählt.

Zeile 17 Hier wird die Identität angegeben, die dieser Client beim Aufbau einer EAP-Verbindung nutzen soll. Falls wir einen Kommunikationspartner hätten, welcher eine spezifische EAP-Identität verlangt, müsste diese hier angegeben werden.

Zeile 18 Der Client gibt (ebenfalls) die Verwendung von EAP als weiteres Authentifizierungsverfahren an. Damit wird die Forderung des RaspberryPi unter 11.2.2.1 erfüllt und die Zwei-Faktor-Authentifizierung verwendet.

11. Zielverbindung mit Zertifikaten

11.1. Generierung von Zertifikaten

Zur sicheren Authentisierung mit strongSwan ist die Verwendung von Zertifikaten erforderlich. Diese können mit der openSSL-Bibliothek erstellt werden. Hierzu bietet openSSL ein Kommandozeilenprogramm. Die in den folgenden Abschnitten eingebetteten Dateien besitzen das Dateiformat „.pem“. Hierfür besteht keine zwingende Notwendigkeit, da die Dateien jeweils als einfache Textdatei interpretiert werden. Jedoch hat sich die Wahl dieser Endung im Rahmen der Implementierung an einigen Stellen als vorteilhaft erwiesen, beispielsweise bei der Verwendung von Zertifikaten. Daher wird die Nutzung dieser Endung empfohlen, jedoch darauf hingewiesen, dass sich hier ein Unterschied zum Vorgehen in anderen Quellen finden könnte.

11.1.1. Certificate Authority (CA)

Zum Signieren von Nutzerzertifikaten ist, wie aus **??** bekannt, eine sogenannte Certificate Authority notwendig, welche den Kommunikationsteilnehmern die Echtheit eines Zertifikates bestätigt.

RSA-Schlüssel generieren

Zu Beginn generieren wir einen RSA-Schlüssel für unsere Certificate Authority, welcher nachher für die Erstellung des Zertifikates verwendet wird. Hierzu verwenden wir das Tool „genrsa“ aus der vorhin genannten „openSSL“-Bibliothek.

```
$ sudo openssl genrsa -out ca.key.pem 2048
```

Signiertes Zertifikat erstellen

Da wir die Erstellung einer PKI anstreben und somit noch keine CA haben, die unsere Zertifikate signieren könnte, sind wir dazu gezwungen, dieses selbst zu signieren. Dies lässt sich mit folgendem Befehl realisieren:

```
$ sudo openssl req -new -x509 -key ca.key.pem -out ca.crt.pem
```

Wie Sie sehen können verwenden wir hier den in 11.1.1 Schlüssel zur Erstellung des Zertifikates.

11.1.2. Nutzerzertifikate erstellen

RSA-Schlüssel generieren

Auch der Nutzer sollte sich auf seinem eigenen Gerät einen RSA-Schlüssel erstellen.

```
$ sudo openssl genrsa -out htwsaariot.key.pem 2048
```

Certificate Signing Request (CSR) erstellen

Dieser Zwischenschritt ist in der Erstellung der Nutzerzertifikate vorgesehen und ermöglicht das systematische Erstellen einer Anfrage zur Erstellung von Zertifikaten.
Hierbei wird, gemeinsam mit dem Schlüssel des Nutzers, eine Datei erstellt, welcher alle notwendigen Informationen des Zertfikatinhabers erhält. Hierbei können diese Daten auch mit Hilfe einer Konfigurationsdatei befüllt werden. Somit wäre, nach einmaliger Erstellung dieser Konfiguration, kein weiterer Eingriff des Nutzers mehr notwendig. Vorteilhaft ist dies bei der Implementierung von PKIs in großen Unternehmen und Organisationsstrukturen.
Erstellt wird eine solche Anfrage mit folgendem Befehl

```
$ sudo openssl req -new -key htwsaariot.key.pem -out htwsaariot.
   csr
```

Wichtige Information Es ist essentiell, dass der Common Name (CN), welcher für den RaspberryPi bzw. den verbindungstragenden Server, die in der „ipsec.conf“ 9.2.3 angegebene rightid enthält. In unserem Fall müsste hier also „htwsaariot.ddns.net“ als CN eingetragen werden. Nur so ist eine Authentifizierung mit „strongSwan“ möglich, da ansonsten die Id innerhalb der Verbindung nicht dem Zertifikat zugeordnet werden kann.

Die Dateiendung „.csr“ deutet bereits darauf hin, dass es sich hierbei um eine Anfrage handelt, welche von der CA bearbeitet werden soll.

Zertifikat signieren (lassen)

Im abschließenden Schritt wird der Signing Request durch die CA signiert. Folgender Kommandozeilenbefehl kann hierfür verwendet werden:

```
$ sudo openssl x509 -req -in htwsaariot.csr
    -CA ca.crt.pem -CAkey ca.key.pem -CAcreateserial
    -out htwsaariot.crt.pem
```

Hier lässt sich erkennen, dass die Bibliothek ein vollkommen eigenständiges Tool namens „x509“ für diesen Zweck anbietet.
Mit diesen nun erstellen Zertfikaten, eins für den RaspberryPi und jeweils eins für jeden Nutzer bzw. jedes Gerät, lässt sich eine Authentifizierung mit „strongSwan“ durchführen.

11.1.3. Zertfikate platzieren

Zur Nutzung der nun erstellen Zertifikate innerhalb unserer strongSwan Konfiguration müssen wir diese in bestimmte, vordefinierte Ordner kopieren/verschieben. Eine Erläuterung zu diesen Ordnern findet sich später in 9.2.1.

Auf dem Server

1. DasZertifikat der „Certificate Authority“ wird unter „/etc/ipsec.d/cacerts/“ kopiert. In unserem Beispiel unter 11.1.1 heißt dieses „ca.crt.pem“.

2. Das Zertifikat unseres Rechners, also hier dem RaspberryPi, welcher hier auch unser Server ist, wird in den Ordner „/etc/ipsec.d/certs“ kopiert.

3. Der private Schlüssel unseres Rechners, erstellt unter 11.1.2, wird nach „/etc/ipsec.d/private“ kopiert.

Die Schritte können analog auf allen weiteren Clients durchgeführt werden. Pro Client sollte folgendes getan werden:

Durchführung pro Client

1. DasZertifikat der Certificate Authority wird unter „/etc/ipsec.d/cacerts/“ kopiert. In unserem Beispiel heißt dieses „ca.crt.pem“.
2. Das Zertifikat unseres Rechners wird in den Ordner „/etc/ipsec.d/certs“ kopiert.
3. Der private Schlüssel unseres Rechners wird nach „/etc/ipsec.d/private“ kopiert.

11.2. Änderung in der strongSwan Konfiguration

Zur leichteren Verständnis sind die bereits in 10.2 aufgeführten Konfigurationsvariablen, die sich mit der hier verwendeten Konfiguration überschneiden, erneut enthalten.

11.2.1. ipsec.secrets

Wir erweitern die in 10.1 gezeigte „ipsec.secrets“ um folgende Werte:

```
1 include /var/lib/strongswan/ipsec.secrets.inc
2
3     : PSK "our-pre-shared-key"
4
5 pi  : EAP "our-eap-password"
6
7     : RSA raspberry.key
```

Zeile 5 definiert einen Nutzernamen sowie ein Passwort für den Verbindungsaufbau mittels EAP.

Zeile 7 Der hier angegebene Wert ist essentiell zur Nutzung unserer Public Key Infrastructure. Hier wird der öffentliche Schlüssel des Nutzers definiert, welcher seinen Schlüssel zur Erstellung der Zertifikatsanfrage- und erstellung verwendet hat. Dabei deutet das Schlüsselwort „RSA“ auf die Art des Schlüssels hin.

11.2.2. Inhalt der ipsec.conf

Die Unterschiede zur in 10.2 gezeigten „ipsec.conf“ Dateien sind nach dem Quellcode aufgeführt. Die Konfigurationsdateien der Klient- uns Serverseite wurden um eine Verbindung „htw-cert“ erweitert, die von den vorherigen Verbindungen (siehe 10.2) erbt und gegebenenfalls Werte überschreibt.

11.2.2.1. Server-Seite

```
# ipsec.conf - strongSwan IPsec configuration file

ca strongSwan
    cacert=ca_cert_v6.pem

config setup
    uniqueids = no

conn %default
    mobike=yes
    dpdaction=clear
    dpddelay=35s
    dpdtimeout=200s
    fragmentation=yes

conn htw
    auto=add
    keyexchange=ikev2
    eap_identity=%any # accepts every user with right password
    left=%htwsaariot.ddns.net # identifies as htwsaariot.ddns.net
    leftsubnet=%any
    leftid=@htwsaariot.ddns.net
    leftauth=psk # authentifies via pre-shared key (see ipsec.
       secrets)
    right=%any
    rightsourceip=%dhcp
    rightauth=eap-mschapv2 # remote host authentifies via eap
    rightdns=8.8.8.8,8.8.4.4 # dns-servers for the remote host
    rightid=%any
    leftfirewall=yes

conn htw-enc
    also="htw"
    ike=aes256-sha2_256-modp1024!
    esp=aes256-sha2_256!

conn htw-cert
    also="htw-enc"
    leftcert=raspi_cert_v6.pem # cert for this client
    leftauth=pubkey # we authenticate via
    rightauth=pubkey
    rightauth2=eap-mschapv2

include /var/lib/strongswan/ipsec.conf.inc
```

„Angabe unserer Certificate Authority“ Mit dem „ca“-Schlüsselwort können wir unsere „Certificate Authority“ bei der eventuellen Verwendung eines Zertifikat-Verfahrens spezifizieren. In Zeile 3-4 geben wir dieser „ca“-Konfiguration einen, an dieser Stelle beliebigen, Namen und spezifizieren das Zertifikat, welches genutzt werden soll. „
Anmerkung: Das zu verwendende Zertifikat muss unter ‚/etc/ipsec.d/cacerts/‘ zu finden sein.“

„Erweiterung um Zertifikate mit ‚htw-cert‘“

Zeile 37 Auch hier werden alle Parameter der zuvor definierten Connection übernommen. Da jedoch in „htw-enc“ auch ein Befehl zur Inklusion von „htw“ steht, werden beide entsprechend verwendet. Die letzte Einstellung eines Parameters ist nachher für den Verbindungsaufbau bindend.

Zeile 38 Hier wird das zu verwendende Zertifikat für die linke Seite aufgezeigt.
„Wichtig: Die Datei muss entsprechend in /‚etc/ipsec.d/certs/‘ vorhanden sein.“

Zeile 39 Mit diesem Parameter wird festgelegt, dass sich unsere Seite mit einem „Public Key“-Verfahren authentifiziert.

Zeile 40 Es wird verlangt, dass unser entfernter Partner ebenfalls mit einem „Public Key“-Verfahren authentifiziert

Zeile 41 Zusätzlich verlangen wir von unseren Verbindungspartnern eine Authentifizierung per EAP. Damit haben wir erfolgreich eine Zwei-Faktor-Authentifizierung implementiert.

11.2.2.2. Client-Seite

```
1  # ipsec.conf - strongSwan IPsec configuration file
2
3  ca strongSwan
4      cacert=ca_cert_v6.pem
5
6  conn %default
7      mobike=yes
8      dpdaction=clear
9      dpddelay=35s
10     dpdtimeout=200s
11     fragmentation=yes
12
13 conn htw
14     right=htwsaariot.ddns.net
15     rightid=%any
16     rightsubnet=0.0.0.0/0
17     rightauth=psk
18     leftsourceip=%config
19     eap_identity=pi
20     auto=add
21
22 conn htw-enc
23     also="htw"
24     ike=aes256-sha2_256-modp1024!
25     esp=aes256-sha2_256!
26
27 conn htw-cert
28     also="htw-enc"
29     leftcert=yta_cert_v6.pem
30     leftauth=pubkey
31     rightauth=pubkey
32     eap_identity=pi
33     leftauth2=eap-mschapv2
34     leftdns=8.8.8.8,8.8.4.4
35
36 include /var/lib/strongswan/ipsec.conf.inc
```

Hier sind ebenso nur solche Zeilen aufgeführt, die sich von der Server Seite sowie der in 10.2.2 aufgeführten Konfigurationsdatei unterscheiden.

Zeilen 3-4 Auch auf dieser Seite geben wir unsere Certificate Authority an, da mit diesem Zertifikat die Gültigkeit der Zertifikate der jeweils anderen Seite sichergestellt werden kann.

Zeile 29 Analog wird hier das zu verwendende Zertifikat unseres Clients angegeben.

12. Nutzung der IPsec Verbindung

12.1. Bedienung des VPN über die Kommandozeile

Die eigentliche Nutzung von strongSwan gestaltet sich nach der initialen Konfiguration vergleichsweise einfach.
Auf allen nutzenden Geräten und vor allem auf dem Server, welcher die Verbindungsanfrage annimmt, sollte der „ipsec“-Daemon laufen. Dieser kann mit folgendem Befehl gestartet werden:

```
$ sudo ipsec start
```

Um einen sauberen Verbindungsaufbau zu gewährleisten, wird desweiteren zu folgendem Befehl geraten

```
$ sudo ipsec restart
```

Dieser startet einen Daemon, falls aktuell keiner lief, und setzt einen bereits in Betrieb genommen Daemon zurück. Um nun eine Verbindung aufzubauen, oder aber auch zu beenden, ist der Verbindungsname essentiell. Denn hier werden, wie bereits erwähnt, alle Parameter angegeben, welche zur Bestimmung der Verbindungspartner und Arten beitragen. Soll nun die Verbindung htw-cert gestartet werden, welche die anderen beiden Konfiguration htw und htw-enc erweitert um die Authentifizierung per Zertifikate, muss folgender Befehl eingegeben werden:

```
$ sudo ipsec up htw-cert
```

Analog kann die Verbindung beendet:

```
$ sudo ipsec down htw-cert
```

12.2. Erklärter Verbindungsaufbau-Log

Beim Aufbau einer Verbindung erscheint folgende Ausgabe in der Kommandozeile „(Anmerkung: Einige Zeilen wurden entfernt, da sie nicht weiter erläutert werden oder zum Verständnis beitragen)“
Eine Erläuterung zu ausgewählten Zeilen findet sich weiter unten.

```
1  initiating IKE_SA htw-cert[3] to 88.68.82.144
2  generating IKE_SA_INIT request 0 [ SA KE No N(NATD_S_IP) N(NATD_D_IP) N(FRAG_SUP)
       N(HASH_ALG) N(REDIR_SUP) ]
3  sending packet: from 192.168.178.99[500] to 88.68.82.144[500] (336 bytes)
4  received packet: from 88.68.82.144[500] to 192.168.178.99[500] (421 bytes)
5  parsed IKE_SA_INIT response 0 [ SA KE No N(NATD_S_IP) N(NATD_D_IP) CERTREQ N(
       FRAG_SUP) N(HASH_ALG) N(MULT_AUTH) ]
6  selected proposal: IKE:AES_CBC_256/HMAC_SHA2_256_128/PRF_HMAC_SHA2_256/MODP_1024
7  {...}
8  received cert request for "C=DE, ST=SL, L=SB, O=HTW, OU=HTW, CN=htwsaariot.ddns.
       net, E=ytatar@htwsaar.de"
9  received cert request for "C=DE, ST=SL, L=SB, O=HTW, OU=HTW, CN=HTW, E=
       ytatar@htwsaar.de"
10 received cert request for "C=DE, ST=SL, L=SB, O=htw, OU=iot, CN=strongSwan, E=
       test@example.org"
11 received cert request for "C=DE, ST=SL, L=SB, O=HTW, OU=HTW, CN=strongSwan, E=
       ytatar@htwsaar.de"
12 sending cert request for "C=DE, ST=SL, L=SB, O=HTW, OU=HTW, CN=strongSwan, E=
       ytatar@htwsaar.de"
13 sending cert request for "C=DE, ST=SL, L=SB, O=htw, OU=iot, CN=strongSwan, E=
       test@example.org"
14 sending cert request for "C=DE, ST=SL, L=SB, O=HTW, OU=HTW, CN=htwsaariot.ddns.
       net, E=ytatar@htwsaar.de"
15 sending cert request for "C=DE, ST=SL, L=SB, O=HTW, OU=HTW, CN=HTW, E=
       ytatar@htwsaar.de"
16 authentication of 'C=DE, ST=SL, L=SB, O=htw, OU=iot g, CN=Yasin Tatar, E=
       ytatar9@googlemail.com' (myself) with RSA_EMSA_PKCS1_SHA2_256 successful
17 sending end entity cert "C=DE, ST=SL, L=SB, O=htw, OU=iot g, CN=Yasin Tatar, E=
       ytatar9@googlemail.com"
18 establishing CHILD_SA htw-cert{9}
19 generating IKE_AUTH request 1 [ IDi CERT CERTREQ AUTH CPRQ(ADDR DNS DNS DNS) N(
       ESP_TFC_PAD_N) SA TSi TSr N(MOBIKE_SUP) N(ADD_6_ADDR) N(ADD_6_ADDR) N(
       MULT_AUTH) N(EAP_ONLY) N(MSG_ID_SYN_SUP) N(AUTH_FOLLOWS) ]
20 splitting IKE message (1696 bytes) into 2 fragments
21 generating IKE_AUTH request 1 [ EF(1/2) ]
22 generating IKE_AUTH request 1 [ EF(2/2) ]
23 sending packet: from 192.168.178.99[4500] to 88.68.82.144[4500] (1236 bytes)
24 sending packet: from 192.168.178.99[4500] to 88.68.82.144[4500] (532 bytes)
25 received packet: from 88.68.82.144[4500] to 192.168.178.99[4500] (1236 bytes)
26 parsed IKE_AUTH response 1 [ EF(1/2) ]
27 received fragment #1 of 2, waiting for complete IKE message
28 received packet: from 88.68.82.144[4500] to 192.168.178.99[4500] (228 bytes)
29 parsed IKE_AUTH response 1 [ EF(2/2) ]
30 received fragment #2 of 2, reassembled fragmented IKE message (1392 bytes)
31 parsed IKE_AUTH response 1 [ IDr CERT AUTH ]
32 received end entity cert "C=DE, ST=SL, L=SB, O=htw, OU=server, CN=htwsaariot.ddns
       .net, E=ytatar@htwsaar.de"
33 using certificate "C=DE, ST=SL, L=SB, O=htw, OU=server, CN=htwsaariot.ddns.net, E
       =ytatar@htwsaar.de"
34 using trusted ca certificate "C=DE, ST=SL, L=SB, O=htw, OU=iot, CN=strongSwan, E=
       test@example.org"
35 checking certificate status of "C=DE, ST=SL, L=SB, O=htw, OU=server, CN=
       htwsaariot.ddns.net, E=ytatar@htwsaar.de"
36 {...}
37 authentication of 'C=DE, ST=SL, L=SB, O=htw, OU=server, CN=htwsaariot.ddns.net, E
       =ytatar@htwsaar.de' with RSA_EMSA_PKCS1_SHA2_256 successful
38 generating IKE_AUTH request 2 [ IDi ]
39 sending packet: from 192.168.178.99[4500] to 88.68.82.144[4500] (80 bytes)
```

```
40 received packet: from 88.68.82.144[4500] to 192.168.178.99[4500] (80 bytes)
41 parsed IKE_AUTH response 2 [ EAP/REQ/ID ]
42 server requested EAP_IDENTITY (id 0x00), sending 'pi'
43 generating IKE_AUTH request 3 [ EAP/RES/ID ]
44 sending packet: from 192.168.178.99[4500] to 88.68.82.144[4500] (80 bytes)
45 received packet: from 88.68.82.144[4500] to 192.168.178.99[4500] (112 bytes)
46 parsed IKE_AUTH response 3 [ EAP/REQ/MSCHAPV2 ]
47 server requested EAP_MSCHAPV2 authentication (id 0xBA)
48 generating IKE_AUTH request 4 [ EAP/RES/MSCHAPV2 ]
49 sending packet: from 192.168.178.99[4500] to 88.68.82.144[4500] (144 bytes)
50 received packet: from 88.68.82.144[4500] to 192.168.178.99[4500] (144 bytes)
51 parsed IKE_AUTH response 4 [ EAP/REQ/MSCHAPV2 ]
52 EAP-MS-CHAPv2 succeeded: 'Welcome2strongSwan'
53 generating IKE_AUTH request 5 [ EAP/RES/MSCHAPV2 ]
54 sending packet: from 192.168.178.99[4500] to 88.68.82.144[4500] (80 bytes)
55 received packet: from 88.68.82.144[4500] to 192.168.178.99[4500] (80 bytes)
56 parsed IKE_AUTH response 5 [ EAP/SUCC ]
57 EAP method EAP_MSCHAPV2 succeeded, MSK established
58 authentication of '192.168.178.99' (myself) with EAP
59 generating IKE_AUTH request 6 [ AUTH ]
60 sending packet: from 192.168.178.99[4500] to 88.68.82.144[4500] (112 bytes)
61 received packet: from 88.68.82.144[4500] to 192.168.178.99[4500] (304 bytes)
62 parsed IKE_AUTH response 6 [ AUTH CPRP(ADDR DNS DNS DNS DNS) SA TSi TSr N(
       AUTH_LFT) N(MOBIKE_SUP) N(ADD_6_ADDR) ]
63 authentication of 'C=DE, ST=SL, L=SB, O=htw, OU=server, CN=htwsaariot.ddns.net, E
       =ytatar@htwsaar.de' with EAP successful
64 IKE_SA htw-cert[3] established between
       192.168.178.99[192.168.178.99]...88.68.82.144[C=DE, ST=SL, L=SB, O=htw, OU=
       server, CN=htwsaariot.ddns.net, E=ytatar@htwsaar.de]
65 scheduling reauthentication in 9917s
66 maximum IKE_SA lifetime 10457s
67 installing 8.8.8.8 as DNS server
68 installing 8.8.8.8 as DNS server
69 installing 8.8.4.4 as DNS server
70 installing 192.168.178.1 as DNS server
71 installing new virtual IP 192.168.178.173
72 {...}
73 selected proposal: ESP:AES_CBC_256/HMAC_SHA2_256_128/NO_EXT_SEQ
74 CHILD_SA htw-cert{9} established with SPIs ce4109d1_i ce263fcb_o and TS
       192.168.178.173/32 === 0.0.0.0/0
75 connection 'htw-cert' established successfully
```

Zeile 1 Es wird der Verbindungsaufbau zum Server initiiert, welche die hier angegebene IP-Adresse 88.68.82.144 hat.

Zeile 3 Pakete werden von 192.168.178.99, also dem aktuell verwendeten Rechner mit seiner Heimnetzadresse, an unseren Server gesendet.

Zeile 6 Zeigt, dass zur Verschlüsselung AES-256 im CBC-Mode verwendet wird oder alternativ HMAC.

Zeile 20 DIe Nachricht wird auf mehrere Fragmente, in diesem Fall 2, aufgeteilt.

Zeile 30 Diese Fragmente werden daraufhin wieder zusammengesetzt

Zeile 32 Das ist das Zertifikat des RaspberryPi, zu erkennen an OU='server' (wurde von uns so eingetragen bei der Zertifikatsgenerierung)

Zeile 37 Der RaspberryPi wurde erfolgreich authentifiziert

Zeile 52 Das zweite von uns angegebene Authentifizierungsverfahren EAP-MSCHAPV2 wurde erfolgreich durchgeführt

Zeile 67 Die DNS-Server für weitere Anfragen werden installiert. Hier können auch eigene Router oder Geräte angegeben werden um zu verhindern, dass der Host externe DNS-Server kontaktiert für Anfragen.

Zeile 71 Das ist die neue IP-Adresse die unser Host im Netzwerk des RaspberryPi erhält.

Zeile 75 Bestätigung das die Verbindung erfolgreich hergestellt wurde.

13. Schlusswort

Durch die Verwendung von „State-of-the-Art“ Technologien, wie der Verwendung von strongSwan mit Zertifikaten und der Einhaltung der vom BSI empfohlenen Standards, konnten wir eine Verbindung in unser Heimnetz aufbauen, die, vergleichbar mit einem SSH-Zugriff oder VPN Verbindungen, die nicht auf Zertifikaten aufbauen, sehr sicher ist.

Man muss sich jedoch nicht in der privaten Nutzung von strongSwan beschränken. strongSwan ermöglicht darüber hinaus beispielsweise, in Kombination mit einer vertrauenswürdigen Zertifizierungsstelle, Mitarbeiter einer Firma, die auf hohe Sicherheitsstandards angewiesen ist, von jedem beliebigen Ort auf das firmeninterne Netzwerk zugreifen zu lassen.
Die mangelnde Dokumentation über das Tool strongSwan erschwerte unsere Nachforschungen. Das Problem bei den verwendeten Technologien sind die Komplexität und der enorme Arbeitsaufwand der Implementierung. Die Nutzung und Konfiguration von strongSwan erfordert besondere Fachkenntnisse und ist somit für einen Normalverbraucher unseres Erachtens nach nicht realistisch.

Eine sicherere Alternative, um einen autorisierten Zugriff in ein Netzwerk zu erlangen, existiert bei korrekter Konfiguration allerdings nicht. Dennoch ist es unserer Meinung nach essenziell, in Zukunft einen höheren Fokus auf die benutzerfreundliche Gestaltung von strongSwan und anderen Sicherheitswerkzeugen zu setzen.

Literatur

[AB04] Bernard Aboba und Larry Blunk. *RFC 3748 - Extensible Authentication Protocol (EAP)*. Mai 2004. URL: https://tools.ietf.org/html/rfc3748 (besucht am 24. 08. 2019).

[AD04] B. Aboba und W. Dixon. *RFC 3715 - IPsec-Network Address Translation (NAT) Compatibility Requirements*. Apr. 2004. URL: https://tools.ietf.org/html/rfc3715 (besucht am 27. 08. 2019).

[And11] Jason Andress. *The Basics of Information Security - Understanding the Fundamentals of InfoSec in Theory and Practice*. 2011.

[AS08] Bernard Aboba und D. Simon. *RFC 5247 - Extensible Authentication Protocol (EAP) Key Management Framework*. Aug. 2008. URL: https://tools.ietf.org/html/rfc5247 (besucht am 24. 08. 2019).

[Blo+02] Marcus Blomenkamp u. a. *Kryptographie und sichere Kommunikation in heterogenen Netzwerken: IPSec*. 2002. URL: https://www.techfak.uni-bielefeld.de/{~}walter/vpn/index.html (besucht am 25. 07. 2019).

[CAP18] CAPEC. *CAPEC-90: Reflection Attack in Authentication Protocol*. 2018. URL: https://capec.mitre.org/data/definitions/90.html (besucht am 27. 08. 2019).

[CIS13] CISCO. *Swift Migration of IKEv1 to IKEv2 L2L Tunnel Configuration on ASA 8.4 Code*. Feb. 2013. URL: https://www.cisco.com/c/en/us/support/docs/security/asa-5500-x-series-next-generation-firewalls/113597-ptn-113597.html#topic1N400024 (besucht am 27. 08. 2019).

[Coo+08] David Cooper u. a. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. en. Mai 2008. URL: https://tools.ietf.org/html/rfc5280 (besucht am 24. 08. 2019).

[ETS10] P. Eronen, H. Tschofenig und Y. Sheffer. *RFC 5996 - An Extension for EAP-Only Authentication in IKEv2*. Sep. 2010. URL: https://tools.ietf.org/html/rfc5996 (besucht am 27. 08. 2019).

[FS00] Niels Ferguson und Bruce Schneier. *A Cryptographic Evaluation of IPsec*. Techn. Ber. 2000. URL: https://www.schneier.com/academic/paperfiles/paper-ipsec.pdf (besucht am 27. 08. 2019).

[GNU19] GNU. *Debian – Das universelle Betriebssystem*. 2019. URL: https://www.debian.org/index.de.html (besucht am 01. 06. 2019).

[HC98] C. Harkins und D. Carrel. *RFC 2409 - The Internet Key Exchange (IKE)*. Nov. 1998. URL: https://tools.ietf.org/html/rfc2409 (besucht am 27. 08. 2019).

[Hou+02] R. Housley u. a. *RFC 3280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. Apr. 2002. URL: https://tools.ietf.org/html/rfc3280 (besucht am 27. 08. 2019).

[Hut+05] A. Huttunen u. a. *RFC 3984 - UDP Encapsulations of IPsec ESP Packets*. Jan. 2005. URL: https://tools.ietf.org/html/rfc3984 (besucht am 24. 08. 2019).

[IEE07] IEEE. *802.11i 2007 - IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. 2007. URL: https://standards.ieee.org/standard/802_11-2007.html.

[Inc18] Dark Cubed Inc. *THE STATEOF IOTSECURITY - It is time for action.* Techn. Ber. 2018.

[KA98a] S. Kent und R. Atikinson. *RFC 2401 - Security Architecture for the Internet Protocol - Updated*. Dez. 1998. URL: https://tools.ietf.org/html/rfc2401 (besucht am 24. 08. 2019).

[KA98b] S. Kent und R. Atkinson. *RFC 2402 - IP Authentication Header*. Nov. 1998. URL: https://tools.ietf.org/html/rfc2402 (besucht am 24. 08. 2019).

[KA98c] S. Kent und R. Atkinson. *RFC 2406 - IP Encapsulating Security Payload*. Nov. 1998. URL: https://tools.ietf.org/html/rfc2406 (besucht am 24. 08. 2019).

[Kal00] Burt Kaliski. *RFC 2898 - PKCS #5: Password-Based Cryptography Specification Version 2.0*. Sep. 2000. URL: https://tools.ietf.org/html/rfc2898 (besucht am 24. 08. 2019).

[Kau98] C. Kaufman. *RFC 4306 - Internet Key Exchange (IKEv2) Protocol*. Dez. 1998. URL: https://tools.ietf.org/html/rfc4306 (besucht am 27. 08. 2019).

[KCB97] Hugo Krawczyk, Ran Canetti und Mihir Bellare. *HMAC: Keyed-Hashing for Message Authentication*. en. Feb. 1997. URL: https://tools.ietf.org/html/rfc2104 (besucht am 24. 08. 2019).

[Kle19] M. Kleine. *Das OSI-Referenzmodell*. 2019. URL: https://www.selflinux.de/selflinux/html/osi.html (besucht am 27. 08. 2019).

[Kra96] H. Krawczyk. *SSKEME: A Versatile Secure Key Exchange Mechanism for Internet, In IEEE Proceedings of the 1996 Symposium on Network and Distributed Systems Security"*. Techn. Ber. 1996.

[KS05] S. Kent und K. Seo. *RFC 4301 - Security Architecture for the Internet Protocol - Updated*. Dez. 2005. URL: https://tools.ietf.org/html/rfc4301 (besucht am 24. 08. 2019).

[Kuh+01] D R Kuhn u. a. *Introduction to public key technology and the federal PKI infrastructure*. en. Techn. Ber. NIST SP 800-32. Gaithersburg, MD: National Institute of Standards und Technology, 2001, NIST SP 800–32. DOI: 10.6028/NIST.SP.800-32. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-32.pdf (besucht am 24. 08. 2019).

[Mas02] Andrew Mason. „IPSec Overview Part Five: Security Associations“. In: (Feb. 2002). URL: http://www.ciscopress.com/articles/article.asp?p=25443 (besucht am 27. 08. 2019).

[Mau+98] D. Maughan u. a. *RFC 2408 - Internet Security Association and Key Management Protocol (ISAKMP)*. Nov. 1998. URL: https://tools.ietf.org/html/rfc2408 (besucht am 26. 08. 2019).

[MVO96] Alfred J. Menezes, Scott A. Vanstone und Paul C. Van Oorschot. *Handbook of Applied Cryptography*. 1st. Boca Raton, FL, USA: CRC Press, Inc., 1996. ISBN: 978-0-8493-8523-0.

[ONe16] Maire ONeill. „Insecurity by Design: Todays IoT Device Security Problem“. In: *Engineering* 2 (März 2016). DOI: 10.1016J.ENG.2016.01.014.

[Ope19] OpenBSD Foundation. *OpenSSH Project History*. Juni 2019. URL: https://www.openssh.com/history.html (besucht am 24. 08. 2019).

[Orm98] H. Orman. *RFC 2412 - The OAKLEY Key Determination Protocol*. Nov. 1998. URL: https://tools.ietf.org/html/rfc2412 (besucht am 27. 08. 2019).

[Pip98] D. Piper. *RFC 2407 - The Internet IP Security Domain of Interpretation for ISAKMP*. Nov. 1998. URL: https://tools.ietf.org/html/rfc2407 (besucht am 27. 08. 2019).

[ST10] Y. Sheffer und H. Tschofenig. *RFC 5273 - Internet Key Exchange Protocol Version 2 (IKEv2) Session Resumption*. Jan. 2010. URL: https://tools.ietf.org/html/rfc5273 (besucht am 27. 08. 2019).

[Sta16] Statista. *Internet of Things - number of connected devices worldwide 2015-2025*. 2016. URL: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/ (besucht am 26. 08. 2019).

[Str18] StrongSwan. *Psec VPN for Linux, Android, FreeBSD, Mac OS X, Windows*. 2018. URL: https://www.strongswan.org/ (besucht am 01. 06. 2019).

[TK08] Hannes Tschofenig und Dirk Kroeselberg. *RFC 5106 - The Extensible Authentication Protocol-Internet Key Exchange Protocol version 2 (EAP-IKEv2) Method*. Aug. 2008. URL: https://tools.ietf.org/html/rfc5106 (besucht am 24. 08. 2019).

[YL06] T. Ylonen und C. Lonvick. *RFC 4251 - The Secure Shell Protocol Architecture*. Jan. 2006. URL: https://tools.ietf.org/html/rfc4251 (besucht am 24. 08. 2019).