Fakultät für
Informatik und
Mathematik

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN
MÜNCHEN

Sichere Verteilung von X.509-Zertifikaten auf Linux-basierten Endbenutzersystemen

Richard Reik

Bachelorarbeit Informatik

Prüfer:
Prof. Dr.-Ing. Thomas Schreck, Hochschule München

10.09.2021

Erklärung

Richard Reik, geb. 27.08.1998 (IF8, SS 2021, Matrikelnummer: 36328517)

Hiermit erkläre ich, dass ich die Bachelorarbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

München, 10.09.2021

Unterschrift

Abstract

Diese Arbeit widmet sich der Fragestellung, wie X.509-Zertifikate für eindeutig identifizierte Linux-basierte Endbenutzersysteme sicher verteilt und ausgestellt werden können. Dazu wird in der Arbeit erst konzeptuell, dann anhand einer beispielhaften Implementierung besprochen, wie ein solches Verfahren umsetzbar ist. Durch seine vielfach getestete Architektur erlaubt die Verwendung des "Trusted Platform Module" dem Linux-basierten Endbenutzersystem, Zertifikate nicht nur sicher zu verwahren, sondern auch Identitäten zu speichern, die es ermöglichen, dieses System eindeutig zu identifizieren. Das "Automated Certificate Management Environment" ist ein Protokoll, das für eine verwandte Anwendung entwickelt wurde. Diese Arbeit erweitert nun das Protokoll, um ein Verfahren zur Verteilung von X.509-Zertifikaten anhand der im TPM-Chip gespeicherten Identitäten zu ermöglichen. Diese Implementierung wird auf ihre Benutzbarkeit evaluiert und es werden weiterführende Sicherheitsmaßnahmen erörtert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listingverzeichnis

1 Einführung

Zum Einstieg in diese Arbeit soll durch eine Motivation sowie durch eine Beschreibung der Problemstellung dargestellt werden, warum diese Arbeit sinnvoll und nützlich ist. Anschließend werden verwandte Arbeiten besprochen, die sich ähnlichen Problemstellungen gewidmet haben, aber auch dargestellt, warum diese Lösungen für die hier beschriebene Problemstellung nicht ausreichend sind. Abschließend wird der Aufbau der Bachelorarbeit besprochen.

1.1 Motivation

Nicht nur für die Kommunikation im Internet werden Zertifikate zur Prüfung der Authentizität der Kommunikationspartner verwendet. Zertifikate und Schlüssel stellen einen zentralen Bestandteil der Sicherheit in der Informatik dar, deren Verlust schwerwiegende Folgen haben kann. Beispielsweise können Geräte und Nutzer nicht mehr eindeutig identifiziert werden und jede Kommunikation zu und mit diesen wird unsicher. Das erlaubt es Angreifern, sich als normalerweise vertrauenswürdige Kommunikationspartner zu tarnen, um zum Beispiel private Daten abzufragen oder Industriespionage zu betreiben. Dabei wird zwischen der Prüfung von Nutzern und Endgeräten unterschieden. Eine Kommunikation kann von Anfang an abgelehnt werden, wenn festgestellt wird, dass das anfragende Gerät nicht vertrauenswürdig ist. Aus diesem Grund muss eine Möglichkeit geschaffen werden, Zertifikate sicher bereitzustellen und abzuspeichern, ohne dass diese von einem Nutzer manipuliert werden könnten.

1.2 Problemstellung

Es muss sichergestellt werden, dass das Zertifikat nicht missbraucht werden kann, ohne dessen Nutzen einzuschränken. Ist die Verwendung des Zertifikates zu kompliziert, ist das Verfahren nutzlos, da es zu unhandlich und damit unbrauchbar wird. Gleichzeitig muss ein Grad an Sicherheit vorherrschen, der den Missbrauch des Zertifikates extrem

erschwert. Das gilt auch für den initialen Prozess des Anfragens des Zertifikates sowie dessen Aktualisierung. Es muss ein Gleichgewicht zwischen Funktionalität und Sicherheit sowie Verwaltbarkeit geben. Das beginnt vor dem Ausstellen des Zertifikates an und endet erst, wenn das Zertifikat sicher verwahrt wurde. Für Windows gibt es bereits ein vergleichbares Verfahren, eine Umsetzung für Linux-Systeme steht jedoch noch aus. Dieser Aufgabe widmet sich diese Arbeit, wobei nicht das Windows-Projekt kopiert, sondern mithilfe verschiedener Verfahren und Prinzipien ein neues Verfahren entwickelt werden soll. Damit das Zertifikat, welches durch das neue Verfahren erhalten wird, auch einen Nutzen für den Endnutzer des entsprechenden Systems mit sich bringt, sollen X.509-Zertifikate ausgestellt werden. Diese werden unter anderem bei der Kommunikation mit "Transport Layer Security" (TLS) verwendet. Das Ziel dieser Bachelorarbeit ist also die sichere Verteilung von X.509-Zertifikaten auf Linux-basierten Endbenutzersystemen.

1.3 Verwandte Arbeiten

Aus der Forschungsfrage lassen sich drei Fragen ableiten. Erstens: Wie kann das Endbenutzersystem eindeutig identifiziert werden? Zweitens: Wie können die entsprechenden X.509-Zertifikate erstellt und verwaltet werden? Drittens: Wie können die Zertifikate sicher von der Zertifizierungsstelle zu den Endbenutzersystemen gelangen? Zertifikate sicher und automatisiert zu verteilen ist die Aufgabe eines Projektes von 2019. In diesem Jahr wurde von der "Internet Engineering Taskforce" (IETF) ein Dokument mit dem Titel "Automated Certificate Management Environment" (ACME) als Internetstandard veröffentlicht[1]. Dieses behandelt eine automatisierte Prüfung sowie die automatisierte Ausstellung von Zertifikaten für die entsprechenden Systeme. Dieses Dokument bildet eine wichtige Grundlage für diese Arbeit, ist jedoch alleine nicht ausreichend, da Gerätevalidierung nicht möglich ist. Es gibt Erweiterungen dieses Protokolls und eine davon, die Prüfung der IP-Adresse[2], soll hier noch kurz erwähnt werden. IP-Adressen sind geeignet, um Zertifikate an Geräte zu verteilen, jedoch reicht die IP-Adresse alleine nicht aus, um das Gerät eindeutig zu identifizieren. Auch "Google" hat sich dem Problem der eindeutigen Identifikation von Geräten gewidmet. So beschreiben die Autoren in einem Vortrag[3] eine Vorgehensweise, die es erlaubt, unter Verwendung von TPM-Chips Geräte eindeutig zu identifizieren. Diese Vorgehensweise, welche im Projekt "go-attestation"[4] beschrieben wurde, bildet die zweite Grundlage dieser Arbeit. Auf diesen aufbauend soll ein eigenes neues Verfahren entwickelt werden, das zusätzliche Sicherheitsmechanismen für Endsystembenutzer bereitstellt.

1.4 Übersicht über die Bachelorarbeit

Um das Ziel dieser Arbeit, die sichere Verteilung von X.509-Zertifikaten auf Linux-basierten Endbenutzersystemen, zu erreichen, wurde diese Bachelorarbeit in vier Teile eingeteilt. Nach dieser Einführung werden im Grundlagenkapitel die wichtigsten Begriffe, die in dieser Arbeit vorkommen, erklärt, sowie eine Einführung in das ACME-Protokoll gegeben. Im Kapitel 3 wird die theoretische Umsetzung einer neuen Challenge für das ACME-Protokoll besprochen. Zusätzlich wird hier auf die Verwendung verschiedener Schutzmaßnahmen von Gerät und Nutzer eingegangen. Im anschließenden 4. Kapitel wird anhand von praktischen Beispielen erklärt, wie das neue ACME-Protokoll konkret umgesetzt werden kann. Diese Implementierung wird im darauf folgenden Kapitel analysiert und auf Funktionalität sowie Schwachstellen überprüft. Hier wird auch ein kleiner Ausblick darauf gegeben, was anders hätte umgesetzt werden können und welche Vor- und Nachteile daraus entstünden. Abschließend wird in einem Fazit die Arbeit noch einmal zusammengefasst und mögliche nächste Schritte besprochen.

2 | Grundlagen

Die Bachelorarbeit baut auf dem TPM sowie dem ACME-Protokoll auf. In diesem Abschnitt der Arbeit sollen diese beiden Themenbereiche so erklärt werden, dass in den folgenden Kapiteln klar wird, auf welchem Fundament die Arbeit aufgebaut ist. Denn sie stellen die Grundlage für die Identifizierung der Endbenutzersysteme sowie die sichere Erstellung, Verteilung und Verwaltung von X.509-Zertifikaten dar. Zuerst werden grundlegende Funktionalitäten des TPM besprochen, bevor der Ablauf sowie die wichtigsten Konzepte des ACME-Protokolls dargestellt werden.

2.1 Trusted Platform Module

Bei dem **TPM** handelt es sich um ein Modul, mit dem Funktionen der Sicherheit auf einer physischen Ebene umgesetzt werden sollen. Obwohl dieses Modul erst 2009 von der "International Organization for Standardization" (ISO) und der "International Electrotechnical Commission" (IEC) [5] beschrieben wurde, existierten 2011 bereits 300 Millionen dieser Chips [6], eine Zahl, die mit der Ankündigung von Windows, für ihr neues Betriebssystem Windows 11 TPM2.0-Chips als Anforderung zu stellen, wahrscheinlich weiter steigen wird[7]. Die Idee des TPM ist es, die Limitationen und Probleme, die Sicherheitssoftware und standardmäßige Hardware mit sich bringen, zu beheben. Dazu gehört ungeschützter und volatiler Speicher sowie unsichere kryptographische Hardware. Der Aufbau des TPM ist recht komplex und wird in dem offiziellen Dokument der "Trusted Computing Group" auf etwa 300 Seiten zusammengefasst[8]. Da das Modul ein eigenes "Application Programming Interface" (API) zur Verfügung stellt, ist es möglich, mit diesem zu kommunizieren. Gleichzeitig besitzt das Modul zwei für diese Arbeit sehr relevante Komponenten. Zum einen, einen geschützten persistenten Speicher, der es erlaubt, wichtige Informationen sicher abzuspeichern[6]. Zum anderen eine kryptographische Einheit, die Schlüsselpaare erstellen kann [9]. Dabei wird sichergestellt, dass die kryptographischen Verfahren auf Hardware ausgeführt werden, die dafür gemacht ist, sichere Ergebnisse zu liefern. Für Schlüsselpaare, die mithilfe der kryptographischen Einheit erstellt werden, kann sichergestellt werden, dass der private Teil des Schlüssels nach der

Erstellung auf dem TPM direkt in dem gesicherten persistenten Speicher abgelegt wird und das Modul zu keiner Zeit verlässt. Diese Tatsache kann verwendet werden, um das Gerät, welches den TPM-Chip verwendet, an diesem zu identifizieren. Jeder Chip verfügt über einen sogenannten "Endorsement Key" (EK). Dabei handelt es sich in der Regel um drei Teile: einen privaten Schlüssel im gesicherten Speicher, auf den nicht zugegriffen werden kann, einen öffentlichen Schlüssel, auf den zugegriffen werden kann, sowie ein dazugehöriges Zertifikat, das vom Hersteller signiert wurde[10]. An diesem Zertifikat in Verbindung mit dem entsprechenden privaten Schlüssel kann der TPM-Chip eindeutig identifiziert werden.

2.2 Automatic Certificate Management Enviroment

Die **ACME** besteht aus zwei Teilen: erstens einer Instanz, die Zertifikate verwalten kann, einer sogenannten "Certificate Authority" (CA), die Zertifikate erstellen und widerrufen kann, und zweitens einer automatisierten Schnittstelle, die erst prüft, ob Anfragen valide sind, und diese dann an die CA weiter gibt. Dadurch können Zertifikate automatisch angefragt werden. Dazu wird auf dem Gerät, das ein Zertifikat benötigt, ein ACME-Client ausgeführt, der eine Anfrage an den entsprechenden ACME-Server stellt. Dieser prüft dann mit Herausforderungen (Challenges), ob der Client tatsächlich der ist, für den er sich ausgibt. Ist der Client in der Lage, die Challenge zu erfüllen, kann er ein Zertifikat anfragen. Dieses Prinzip soll sich für diese Bachelorarbeit zunutze gemacht werden. Auch hier soll mit einem automatisierten Verfahren erst der Client geprüft und anschließend ein Zertifikat ausgestellt werden. Der ACME-Server kann dabei nur mit integrierter Datenbank verwendet werden, da jeder Client, der ein Zertifikat anfragen möchte, zuerst ein Konto erstellen muss, das in der Datenbank gespeichert wird. Für dieses Konto kann der Client dann Anfragen nach Zertifikaten schicken, muss dabei jedoch für jede Anfrage eine Challenge erfüllen. Die Art der Challenge, ob nun DNS oder HTTP – beide werden im Verlauf der Arbeit noch genauer behandelt – kann der Client dabei frei wählen. Die folgenden Erklärungen beziehen sich auf ACME, wie es im RFC8555 definiert ist [1].

2.2.1 Hintergrund

Ein ACME-Server, der nicht gleichzeitig als Webserver dient, muss laut RFC8555 mindestens eine Schnittstelle zur Verfügung stellen, die unverschlüsselt erreicht werden kann. Diese dient als Übersicht (Directory) über alle URLs die benötigt werden, damit der Client mit dem Server kommunizieren kann. Darunter finden sich zum Beispiel URLs,

um ein Zertifikat zu widerrufen, eine Replay-Nonce zu erhalten und ein Konto zu erstellen. Eine Replay-Nonce ist eine zufällig generierte Zahl, die jede Nachricht eindeutig kennzeichnet und so vor Replay-Angriffen schützt. Jede Kommunikation, mit Ausnahme des GET-Requests zum Erhalten des Directory sowie dem Erhalten der Replay-Nonce, ist verschlüsselt und muss mit einer Replay-Nonce abgesichert werden. Dabei übersendet der Server bei jeder Anfrage des Clients auch eine Replay-Nonce im Header der Antwort mit, sodass diese nicht jedes Mal neu angefragt werden muss. Jede Kommunikation, ausgenommen die genannten zwei, muss als POST oder POST-as-GET-Request durchgeführt werden. POST-as-GET bedeutet, dass jede Anfrage, die normalerweise ein GET-Request wäre, nun als POST-Request mit leerem, aber signiertem Body, geschickt wird. POST-as-GET-Requests sind unabdingbar, da jede Kommunikation durch "JSON Web Signature" (JWS) gesichert werden muss und der Body des GET-Requests keine definierte Form hat[11]. Die Schlüsselpaare, die für die Kommunikation mit JWS notwendig sind, müssen vorher clientseitig erstellt werden.

2.2.2 Ablauf

Im Folgenden soll der Ablauf einer ACME-Kommunikation von der ersten Nachricht bis zum Erhalt des Zertifikates dargestellt werden. Der Ablauf des ACME-Protokolls ändert sich abhängig davon, ob der Client bereits über ein Konto auf dem Server verfügt. Hier soll ein Ablauf beschrieben werden, in dem der Client noch keinerlei Kontakt mit dem Server hatte, also kein Konto besitzt. Der Einfachheit halber wird die Kommunikation aus Sicht des Clients dargestellt und die internen Abläufe des ACME-Servers außer Acht gelassen, da sie von Server zu Server unterschiedlich sein können. Zuallererst muss der Client eine Anfrage an das Directory stellen, um zu erfahren, welche URL für welche Kommunikation benötigt wird. Sind die URLs bereits bekannt, kann dieser vorbereitende Schritt übersprungen werden.

2.2.2.1 Nonce

"Replay"-Noncen werden im ACME-Protokoll verwendet um vor "replay"-Angriffen zu schützen. Dabei muss jede Kommunikation vom Client zum Server durch eine solche Nonce geschützt sein, nur zwei im Kapitel 2.2.1 besprochene Anfragen sind von dieser Regel ausgenommen. Im weiteren Ablauf des Protokolls wird die Nonce in jeder Antwort des Servers mitgeschickt, damit der Client nicht wieder eine neue Anfrage nur für die "Replay"-Nonce senden muss. Dadurch entsteht eine Kette, die aus der Anfrage des Clients mit erhaltener Nonce und Antwort des Servers mit neuer Nonce besteht. Wenn die Nonce jedoch abgelaufen ist, da die letzte Kommunikation länger zurückliegt, oder

der Client noch gar keine Anfrage gestellt hat und somit noch keine Nonce erhalten hat, kann der Client eine neue Nonce anfragen. Dazu schickt der Client einen HEAD-Request an den Server an die über das Directory erhaltene URL. Da jede Kommunikation, die nun beschrieben wird, eine "Replay"-Nonce verwendet, wurde darauf verzichtet, dies immer wieder anzuführen. Wie der HEAD-Request und die entsprechende Antwort dabei aussehen, wird im Listing 2.2.2.1 ("7.2 Getting a Nonce") verdeutlicht. Der erste Teil beschreibt eine Anfrage zum Erhalten der Nonce, der zweite Teil die entsprechende Antwort des Servers.

```
HEAD /acme/new-nonce HTTP/1.1
Host: example.com

HTTP/1.1 200 OK
Replay-Nonce: oFvnlFP1wIhRlYS2jTaXbA
Cache-Control: no-store
Link: <https://example.com/acme/directory>;rel="index"
```

Listing 2.1: '7.2 Getting a Nonce', RFC8555

2.2.2.2 Konto erstellen

Jede Order wird fest an ein Konto gebunden und so muss zu Beginn ein neues Konto erstellt werden. Dazu sendet der Client in seiner "JWS Payload" Mailadressen, die mit diesem Konto verknüpft werden sollen, sowie eine Bestätigung an den Server, dass der Client mit den Nutzungsbedingungen einverstanden ist. Optional kann in diesem Schritt auch ein bereits vorhandenes Konto verknüpft werden. Wie im folgenden Listing zu sehen, wird hier, da noch keine "Key ID" (KID) vorhanden ist, im Header an dessen Stelle der "JSON Web Key" (JWK) mit dem entsprechendem öffentlichen Schlüssel, der zum Signieren verwendet wurde, an den Server gesendet. Dieses Listing beschreibt dabei den Inhalt einer Anfrage zur Erstellung eines Kontos, wie er im RFC8555 definiert wird.

```
"protected": base64url({
  "alg": "ES256",
  "jwk": {...},
  "nonce": "6S8IqOGY7eL2lsGoTZYifg",
  "url": "https://example.com/acme/new-account"
}),
"payload": base64url({
  "termsOfServiceAgreed": true,
  "contact": [
    "mailto:cert-admin@example.org",
    "mailto:admin@example.org"
```

```
  ]
}),
"signature": "RZPOnYoPs1PhjszF...-nh6X1qtOFPB519I"
```

Listing 2.2: '7.3 Account Management' (Teil 1), RFC8555

Die Antwort des Servers wird musterhaft im nächsten Listing dargestellt. Dabei übersendet der Server neben den "contact"-Informationen, die er vom Client erhalten hat, auch den Status sowie die URL für die entsprechende Order mit. Im Header der Antwort übersendet der Server dem Client auch seine Konto-URL, welche im folgenden Ablauf der Client-Server-Kommunikation als KID fungiert.

```
{
  "status": "valid",

  "contact": [
    "mailto:cert-admin@example.org",
    "mailto:admin@example.org"
  ],

  "orders": "https://example.com/acme/acct/evOfKhNU60wg/orders"
}
```

Listing 2.3: '7.3 Account Management' (Teil 2), RFC8555

2.2.2.3 Order platzieren

Mit den im letzten Schritt erhaltenen Informationen kann der Client nun eine Order erstellen. Dazu sendet er in der "payload" ein "identifier"-Array mit allem, was er validiert haben möchte, an den Server. In diesem Array wird nicht nur definiert, mit welchem Verfahren, dem sogenannten "type", die Validierung stattfinden soll, sondern auch gegen welchen Wert ("value"). Der Client kann sich aussuchen, wie er geprüft werden möchte. Die prominentesten zwei Verfahren, die DNS- sowie die HTTP-Challenge, werden im nächsten Schritt näher erläutert. Für beide übersendet der Client im "type" des "identifier" den Wert "dns" mit, wie im Listing 2.2.2.3 zu sehen. Zusätzlich kann der Client durch das Übersenden von "notBefore"- und "notAfter"-Werten bestimmen, für welchen Zeitraum das Zertifikat gültig sein soll, das ist jedoch optional. In der Antwort schickt der Server den Status, wann die Gültigkeit der Anfrage ausläuft, sowie ein Array an Links zur Validierung der Order mit. Für jeden "identifier" mit "type" und "value" wird im sogenannten "authorizations"-Array genau ein Link erstellt. Zusätzlich antwortet der Server mit einer "finalize"-URL, die im späteren Verlauf benötigt wird.

```
{
  "protected": base64url({
    "alg": "ES256",
    "kid": "https://example.com/acme/acct/evOfKhNU60wg",
    "nonce": "5XJ1L3lEkMG7tR6pA00clA",
    "url": "https://example.com/acme/new-order"
  }),
  "payload": base64url({
    "identifiers": [
      { "type": "dns", "value": "www.example.org" },
      { "type": "dns", "value": "example.org" }
    ],
    "notBefore": "2016-01-01T00:04:00+04:00",
    "notAfter": "2016-01-08T00:04:00+04:00"
  }),
  "signature": "H6ZXtGjTZyUnPeKn...wEA4TklBdh3e454g"
}
```

Listing 2.4: '7.4 Applying for Certificate Issuance', RFC8555

2.2.2.4 Challenge aktivieren

Für den Client gibt es mehrere Methoden, mit denen er beim Server validieren kann, dass er tatsächlich die Identität besitzt, die er vorgibt zu haben. Im RFC8555-Dokument werden dabei zwei Methoden näher erläutert, die auch hier ausführlicher besprochen werden sollen, die HTTP- und die DNS-Challenge. Weiterführend gibt es Erweiterungen für das Dokument, wie eine Challenge, welche die Kontrolle über eine IP-Adresse[2], oder die Domain über TLS, prüft [12]. Um zu verstehen, wie ACME funktioniert, reicht es aber aus, sich auf DNS- und HTTP-Challenge zu beschränken.

Der Client sendet nun einen POST-as-GET-Request an den Server unter der URL, deren Challenge er als erstes bearbeiten möchte. Der Server antwortet nun mit Informationen zu dieser Challenge, wie dem Status, wann sie abläuft, den entsprechenden "identifier"-Werten und einem Array mit Challenges. Diese Challenges haben als Wert für den "type" entweder "http-01" oder "dns-01". Beide haben eine eigene URL sowie einen gemeinsamen Token. Der Token ist ein zufälliger base64-Wert, der die Challenge eindeutig identifiziert. Die im Folgenden verwendeten Beispiele stammen aus dem RFC8555-Dokument.

In der DNS-Challenge kann der Client aus diesem Token in Verbindung mit seinem eigenen kontogebundenen Schlüssel (Account Key), einen Autorisierungsschlüssel

(Authorization Key) erstellen. Dieser Schlüssel wird anschließend mit dem SHA-256-Verfahren verschlüsselt (hashed). Der so erhaltene Wert wird nun als base64-Wert in einem "TXT Resource Record" im DNS gespeichert. Dieser Wert wird dabei unter dem Wert, der im "value" des "identifier"-Arrays angegeben wurde, nach dem Präfix "_acme-challenge" abgespeichert. Für "www.example.org" wird der Wert also unter "_acme-challenge.www.example.org" abgespeichert. Der Client sendet nun einen POST-as-GET-Request zum Server, um diesen zu informieren, dass dieser die Information anfragen kann.

Die HTTP-Challenge läuft sehr ähnlich ab. Hier wird genauso ein Autorisierungsschlüssel erstellt, nur wird dieser Schlüssel unter "[domain]/.well-known/acme-challenge/[token]" für einen GET-Request zur Verfügung gestellt.

2.2.2.5 Challenge erfüllen

Der Client sendet nun einen POST-as-GET-Request an den Server, um ihn wissen zu lassen, dass er mit der Validierung beginnen kann. Um zu validieren, dass die Challenge erfüllt wurde, erstellt der Server denselben Hash, fragt von der Domain das "TXT Resource Record" oder die HTTP-Ressource an und überprüft, dass der erhaltene Wert mit dem eigenen Wert übereinstimmt. Konnte der Wert abgefragt werden und stimmt er mit dem erstellten Wert überein, gilt die Challenge als bestanden.

2.2.2.6 Zertifikatsmanagement

Um die entsprechenden URLs, die der Client benötigt, bereitzustellen, erweitert der Server Stück für Stück die Order. Wird der Status dieser Order als "valid" gesetzt, wurden also alle Challenges erfüllt, so fügt der Server der Order eine "finalize"-URL hinzu. Mit dieser URL kann der Client ein Zertifikat anfragen. Dazu sendet er dem Server einen "Certificate Signing Request" (CSR) an die "finalize"-URL. Der Server generiert nun anhand dieses CSR noch einmal die Order und fügt eine "certificate"-URL hinzu. Mithilfe dieser URL teilt der Server dem Client mit, wo das für ihn generierte Zertifikat bereitgestellt wird.

Um das Zertifikat anzufordern, muss der Client jetzt nur noch einen POST-as-GET-Request an die im letzten Schritt mitgeteilte URL senden. Der Server antwortet mit dem Zertifikat im Body der Antwort. Damit ist das Zertifikat erhalten und der Ablauf des Protokolls abgeschlossen.

2.2.2.7 Weitere mögliche Schritte

Der Client kann nun über das erstellte Konto die Zertifikate aktualisieren lassen, ohne die Challenges noch einmal durchlaufen zu müssen. Der Client kann den Server bitten, das Konto zu löschen oder ein Zertifikat zu widerrufen. Auch ein sogenannter "Key Change" ist möglich, wenn der öffentliche und der private Schlüssel, die für das Konto und damit auch für die Kommunikation mit JWS verwendet wurden, geändert werden sollen. Dabei muss der Client den neuen Schlüssel in einer Nachricht verpacken, welche von dem alten Schlüssel signiert wurde.

3 | Theoretische Umsetzung

Da nun die Grundlagen des ACME-Protokolls besprochen wurden, soll nun eine neue Challenge mit all den Änderungen, die diese mit sich bringt, besprochen werden, bevor im nächsten Kapitel deren praktische Umsetzung diskutiert wird. Zunächst soll beschrieben werden, welche Vorbereitungen getroffen werden müssen sowie welche Abläufe gleich bleiben und welche sich verändern sollen. Anschließend wird kurz dargestellt, welche zusätzlichen Schritte unternommen werden sollen, um dem Ziel, das Endbenutzersystem eindeutig identifizieren zu können, zu erreichen. Auch wie der Vorgang server- und clientseitig sicherer gestaltet werden kann, wird kurz besprochen.

3.1 Aufbau der neuen ACME-Challenge

Wie im Grundlagenkapitel beschrieben, ist die Challenge das Herzstück des ACME-Protokolls. Ein Ziel dieser Bachelorarbeit besteht darin, eine neue Challenge zu erstellen. Dabei sollen die gleichen Sicherheitsstandards wie bei jeder anderen ACME-Kommunikation gelten. Um das zu erreichen, werden auch hier POST- und POST-as-GET-Requests verwendet. Mit ihnen in Verbindung mit JWS muss der Client in jeder Anfrage seine Nachrichten mit seinem Account Key signieren. Dadurch kann bei jeder Kommunikation genau bestimmt werden, um welchen Client es sich handelt. Auch Replay-Noncen sollen wieder eingesetzt werden. Sie sollen in jeder Kommunikation des Servers mit dem Client, die über das Anfragen eines Directorys hinausgehen, mitgeschickt werden. Alleine durch diese Maßnahme kann Gefahrenquellen wie Replay-Angriffen entgegengewirkt werden. Der im Grundlagenkapitel beschriebene Aufbau, in dem erst ein Konto erstellt, dann eine Challenge erzeugt, diese dann beantwortet und dann das Zertifikat mithilfe eines CSR erzeugt wird, bleibt also erhalten. Die neue Challenge soll dabei nicht die Kontrolle über eine Webseite oder einen DNS prüfen, sondern ein Endbenutzersystem eindeutig identifizieren können.

3.1.1 Vorbereitung

Im Gegensatz zu den beiden oben besprochenen Challenge-Arten soll die neue Challenge die Kontrolle über das Gerät, für das sich der Client ausgibt, überprüfen. Der erste Schritt besteht darin, entweder vom Hersteller oder per Hand den öffentlichen Schlüssel des "Endorsement Key" (EK) aus dem TPM-Chip zu erhalten. Der sogenannte EK ist ein fester Bestandteil des TPM-Chips und wird bei dessen Erstellung vom Hersteller mit eingebaut. Der private Teil kann dabei von außen weder ausgelesen noch angefragt werden. Diese Tatsache wird verwendet, um das Gerät eindeutig zu identifizieren. Der öffentliche Teil des EK wird serverseitig gespeichert. Da der EK schon früh in der Kommunikation zwischen Server und Client mitgeschickt werden soll, kann der Server abgleichen, ob eine Anfrage tatsächlich von einem Gerät kommt, das auf dem Server registriert ist, und die Kommunikation frühzeitig beenden, falls das nicht der Fall ist.

3.1.2 Konto erstellen und verwalten

Das im Grundlagenkapitel beschriebene Anfragen des Directorys sowie die Verwendung von Replay-Noncen oder das Erstellen des Kontos haben sich weder client- noch serverseitig geändert. Die einzige Änderung, die vorgenommen werden kann, jedoch optional für den weiteren Ablauf der neuen Challenge ist, ist das Erstellen des privaten und öffentlichen Schlüssels des Kontos durch den TPM-Chip. Jedes Konto wird durch seinen öffentlichen Schlüssel beim Server registriert. Dieses Schlüsselpaar wird unter anderem für die Kommunikation mit JWS, also das Signieren der Nachrichten verwendet. Da der TPM-Chip über eine kryptographische Einheit verfügt, kann sich der Client darauf verlassen, dass die so generierten Schlüssel sicher sind. Auf anderer Hardware ist das nicht so konsequent gewährleistet.

3.1.3 Die EK-Challenge

Wie die DNS- und HTTP-Challenge besteht auch die neue EK-Challenge aus zwei Teilen, erst dem Absenden der Order, dann dem Erfüllen der Challenge. Für die Challenge wird clientseitig mithilfe des TPM-Chips ein sogenannter AK erstellt. Der "Attestation Key" (AK), wie er hier verwendet wird, ist ein Container, der neben einem öffentlichen Schlüssel auch Metadaten, wie unter anderem die TPM-Version, besitzt. Der AK dient nicht nur als normaler Schlüssel, sondern vielmehr als eine Verpackung für Informationen über den TPM-Chip, Informationen, die zum Erstellen eines Geheimnisses benötigt werden. Zusätzlich zu diesen Informationen, die der AK beinhaltet, ist es wichtig, diesen zu erstellen, da der EK alleine, abhängig von der TPM-Implementierung, eventuell nicht in

der Lage ist, selbst zu verschlüsseln. Ziel dieser Verbindung aus EK und AK ist es, durch den EK das Gerät zu identifizieren und durch ein Geheimnis, das durch die Verwendung von EK und AK erstellt wird, zu verifizieren. Ein Geheimnis ist dabei nichts anderes als eine verschlüsselte Information, welche nur durch die entsprechenden privaten Schlüssel entschlüsselt werden kann. Das dabei verwendete Verfahren basiert auf einem Projekt von "Google" zur Identifizierung von Geräten [4]. Durch dieses Verfahren kann auch der AK eindeutig dem Chip zugeordnet werden, es wird also auch sichergestellt, dass der AK vom gleichen Chip stammt wie der EK.

Für die Order werden nun AK sowie EK zusammen als "value" für den "identifier" bestimmt, der Type trägt nun den Namen der neuen Challenge "ek". Abgesehen von dieser Änderung wird die Anfrage regulär an den Server gesendet. Dieser kann nun überprüfen, ob der EK-Wert in seiner Datenbank vorkommt. Falls das nicht der Fall ist, wird dem Client ein HTTP-400-Fehler zurückgegeben, kommt der EK-Wert jedoch vor, kann nun mit der eigentlichen Challenge begonnen werden. Dazu erstellt der Server unter Verwendung der AK- und EK-Werte ein Geheimnis.

Der Client kann daraufhin mit einem POST-as-GET-Request dieses Geheimnis erfragen. Das Geheimnis kann nur mithilfe des TPM-Chips entschlüsselt werden, und die so entschlüsselte Nachricht wird wieder zurück an den Server gesendet. Wurde das Geheimnis korrekt entschlüsselt, gilt die Challenge als erfüllt und der Server ändert den Status der Challenge von "pending" über "processing" zu "valid". Durch das Erfüllen der Challenge wird sichergestellt, dass der Client die Kontrolle über den TPM-Chip besitzt. Die Prüfung der Identität des Client-Geräts ist eine Prüfung der Kontrolle über den entsprechenden TPM-Chip.

3.1.4 CSR

Für die Erstellung des CSR verwendet der Client nun den TPM-Chip, zum einen da dieser über eine eingebaute kryptographische Funktion verfügt, zum anderen weil der private Schlüssel nie außerhalb des Chips existieren darf. Aus dem öffentlichen Schlüssel, zusammen mit anderen Daten, die im Zertifikat stehen sollen, erstellt der TPM-Chip eine CSR. Die CSR wird im Body der Nachricht an den Server gesendet, der das Zertifikat erstellt und dem Client zur Verfügung stellt, sodass dieser es per POST-as-GET-Request abfragen kann. Das so erhaltene Zertifikat wird nun auf dem TPM-Chip gespeichert. Der Schlüssel, der zur Erstellung des Zertifikates verwendet wurde, ist dabei der gleiche, der mit dem EK-AK-Verfahren überprüft wurde.

3.1.5 Folgeanfragen

Der Client hat nun dem Server gegenüber bewiesen, dass er tatsächlich die Kontrolle über den TPM-Chip, den er angegeben hat, besitzt. Diese Information ist jedoch nur solange gültig, wie das Zertifikat gültig ist. Möchte der Client ein neues Zertifikat beantragen, so muss er einen neuen AK-Wert generieren, der den gleichen Prozess durchläuft wie der erste AK beim ersten Mal, als eine Order erstellt wurde. Er muss dem Server gegenüber noch einmal beweisen, dass er die Kontrolle über diesen Chip besitzt. Dieses Vorgehen wird so im RFC8555-Dokument festgehalten. Zu diesem Verfahren gibt es eine Alternative, die auch kurz besprochen werden soll. Dabei wird zur Validierung des Clients beim Erstellen einer neuen Order das alte, aber noch gültige Zertifikat verknüpft, das den Client bereits validiert hat. Es handelt sich dabei um die Verkettung von Zertifikaten[13]. Durch die Verknüpfung der Anfrage auf ein neues Zertifikat mit dem alten Zertifikat wird bewiesen, dass der Client die Kontrolle über den Chip immer noch besitzt. Auf die Vor- und Nachteile dieses Verfahrens im Vergleich zum hier gewählten wird in der Evaluation noch ausführlicher eingegangen.

3.2 Zusätzliche Maßnahmen

Für diese Bachelorarbeit wurde neben der Verteilung von Zertifikaten auch ein Augenmerk auf Sicherheit gelegt und geprüft, wie dieses Verfahren sicherer gestaltet werden kann. Das gilt für das ACME-Protokoll, aber auch für die Verwahrung des Zertifikates auf dem Endbenutzersystem des Client sowie eine serverseitige Erweiterung der Datenbank um zusätzliche Möglichkeiten der Account-Verwaltung. Diese Änderungen sind nicht notwendig, um das Ziel dieser Arbeit, die sichere Verteilung von X.509-Zertifikaten auf Linux-basierten Endbenutzersystemen, zu erreichen. Sie stellt jedoch einen möglichen nächsten Schritt für die Sicherheit und Verwendbarkeit des Protokolls dar.

3.2.1 Clientseitig

Der Client verfügt über die Möglichkeit, unter Verwendung des TPM-Chips ein Zertifikat vom spezifizierten ACME-Server zu erhalten. Das gesamte Verfahren wäre nutzlos, wenn es für einen Benutzer des Client-Geräts möglich wäre, wertvolle Informationen aus dem Ablauf des neuen ACME-Verfahrens abzufangen. Es muss also eine zusätzliche Sicherheit geschaffen werden, um das Client-Gerät vor seinen eigenen Nutzern zu schützen. Eine dementsprechende Maßnahme ist, dass der private Schlüssel, welcher zum Zertifikat gehört, den TPM nie verlässt. Aber auch Metadaten könnten für einen Angreifer interessant

sein. Neben böswilligen kann es auch fahrlässige Nutzer geben, die schlicht vergessen, ein Zertifikat anzufragen oder zu erneuern, falls es veraltet. Aus all diesen Gründen wird ein "System Daemon" geschaffen, der die clientseitige Kommunikation mit dem ACME-Server übernimmt. Dabei soll geprüft werden, ob ein Zertifikat vorhanden ist. Falls nicht, wird ein neues erstellt. Ist ein Zertifikat vorhanden, aber veraltet oder läuft bald aus, so wird ein neues Zertifikat angefragt. Dadurch läuft die Kommunikation im Hintergrund ab und ist für den Nutzer des Gerätes unsichtbar. Zertifikate werden dem Nutzer nur durch den TPM-Chip zur Verfügung gestellt.

3.2.2 Serverseitig

Der ACME-Server wird um die Funktionalität der Datenbankeinträge für öffentliche EK-Schlüssel erweitert, was für die Bearbeitung der neuen EK-Challenge notwendig ist. Durch eine Datenbank-API soll es einem Systemadministrator erleichtert werden, die Liste der bekannten öffentlichen EK-Schlüssel zu erweitern. Auch serverseitig gibt es Möglichkeiten, wie der Umgang mit EK-Werten sicherer gestaltet werden kann. So ist es möglich, sobald ein Client eine "ek"-Challenge gestellt und bestanden hat, dessen Konto mit dem entsprechendem EK-Wert in der Datenbank zu verknüpfen. So kann immer eindeutig identifiziert werden, wenn ein neues Zertifikat erstellt wird, für welchen TPM dieses gilt. Damit ist es auch einfacher, Zertifikate zu widerrufen, sollte das Gerät als gestohlen oder vermisst gemeldet werden, da nun dem dazugehörigen Datenbankeintrag, mit all seinen verknüpften Informationen, nicht mehr vertraut werden kann.

4 Praktische Umsetzung

Im folgenden Kapitel soll beschrieben werden, wie unter Zuhilfenahme der Informationen aus den letzten zwei Kapiteln eine Implementierung des Verfahrens realisiert wurde. Dazu wurde dieses Kapitel in zwei Teile eingeteilt. Im ersten Teil soll der Aufbau des Systems aus Linux-basiertem Client mit zugehörigem TPM-Chip und Server mit zugehöriger Datenbank besprochen werden. Im zweiten Teil geht es um die Implementierung des Verfahrens, die anhand von ausgewählten Codebeispielen erklärt werden soll.

4.1 Architektur

Wie in Abbildung 4.1 verdeutlicht werden soll, handelt es sich bei der Lösung um zwei miteinander kommunizierende Parteien, den ACME-Server und den ACME-Client. Bei dem ACME-Server handelt es sich um eine inhaltliche Erweiterung für ein bereits existierenden ACME-Server sowie das Erweitern eines Datenspeichers. Auf der Clientseite steht ein Linux-System, das auf einen TPM-Chip zugreifen kann.

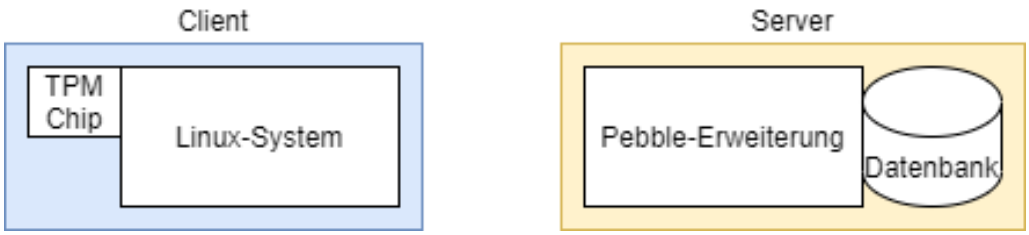

Abbildung 4.1: Vereinfachte Darstellung des Client-Server-Aufbaus

4.1.1 Einrichtung des ACME-Client

Für die Einrichtung des ACME-Client müssen zwei Voraussetzungen erfüllt sein: Erstens, dass der Client auf einem Linux-System läuft, wobei gleichgültig ist, welche Linux-Distribution verwendet wird, zweitens die korrekte Installation des TPM-Chips. Hierbei gibt es zwei Möglichkeiten, wie der Chip installiert werden kann. So ist es möglich, den

Chip an dem jeweiligen Gerät entsprechend der Anleitung physisch zu befestigen und einzurichten. Der Chip kann aber auch auf dem Client-Gerät simuliert werden. Welche Simulation dabei verwendet wird, ist dem Architekten des Systems überlassen. Es muss jedoch beachtet werden, dass die Sicherheit, wie sie in den vorangegangen Kapiteln beschrieben wurde, nur durch den physischen Chip garantiert werden kann. Da der Chip immer nur eine Kommunikation gleichzeitig erlaubt, muss sichergestellt werden, dass kein anderes Programm gerade mit diesem kommuniziert und eine Kommunikation auf diese Weise blockiert. Ein einfacher Test in Linux sieht folgendermaßen aus:

```
$ sudo cat /dev/tpm0
```

Listing 4.1: Schnellcheck für die TPM-Erreichbarkeit

Erhält man auf diese Anfrage die Antwort "resource is busy", wird der TPM-Chip durch einen anderen Prozess blockiert. In der hier beschrieben praktischen Umsetzung wurde ein physischer TPM-Chip und keine Emulation oder Simulation verwendet.

4.1.2 Aufsetzen des ACME-Servers

Für diese Arbeit wurde sich bereits existierender Software bedient. "Pebble" ist ein ACME-Server, der von "letsencrypt" zur Verfügung gestellt wird[14]. Dabei handelt es sich um eine vereinfachte Version, die für Testzwecke, aber nicht auf live-Systemen verwendet werden sollte. Diese stellt alle grundlegenden Funktionen, die für die folgenden Schritte benötigt werden, zur Verfügung. Einzig ein persistenter Datenspeicher muss hierfür erstellt werden. Da es sich um ein Proof-of-Concept handelt, wurden die EK-Werte, gegen die getestet werden soll, statt in einer realen Datenbank in GO-Programmen hartkodiert, wie im folgenden Listing zu sehen ist. Wichtig ist nur, dass die Werte persistent gespeichert sind und der Server diese abfragen kann. Die Datei "wfe.go" stammt von "Pebble" und wurde für das eigene Projekt erweitert[15].

```
const pubPEM = `
-----BEGIN RSA PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAh2oOFWso2nWgrA/6SIcJ
xznL4ZHw1rVnphcqYVChhzC8tXdZ6eZmPWbIP4xgKtZsYSAkPbo1Lf3dPFlA+G5W
xuXpE5QRn1bIo3Rx0CxLwduy/z7Eak8HNI32eb1U2jPYqCMCeLRStNjNnqZEoji4
//cqss1B1pXWJCH8VckfpSiXBvA+0Jyk5ceY83VCVYoKBwLVhRnTEFI2TeWUOFDn
136c85//Yd+Mohx9aoTyYTiC84ePO/sJoNdKaFl8JjgsqxYPFxcCguzeCacvA/Jr
Ps853EG0Sl52FuBj21CeB8QJUrNpabT/kFM9kBW6HQvWEgASvO0FTJ42lCx80Ecv
mQIDAQAB
-----END RSA PUBLIC KEY-----`
```

Listing 4.2: EK öffentlicher Schlüssel, wfe.go

4.2 Implementierung

Das Projekt wurde sowohl server- wie auch clientseitig in der Programmiersprache "GO" geschrieben. Als Server wurde der eigene Rechner verwendet, der Client lief auf einem Raspberry PI, der Code für beide wurde vorrangig auf dem Rechner geschrieben. Die IP-Adressen sind beispielhafte Werte, die nur dazu dienen um aufzuzeigen, wie die Kommunikation zum ACME-Server ermöglicht wird und, darauf aufbauend, wie die neue EK-Challenge implementiert und verwendet werden kann. Die verwendeten Listings sollen dabei als visuelle Stütze dienen und sind nicht ausreichend, um das Projekt nachzubauen. Bei den Dateien "wfe.go" und "common.go" handelt es sich um Erweiterungen des "Pebble" Projekts, also des ACME-Servers[15]. "encryption.go" stellt das funktionale Rückgrat des Clients dar[16].

4.2.1 Implementierung des regulären ACME-Ablaufs

Wie bereits in den Grundlagen beschrieben, bedient sich die Kommunikation zwischen dem Client und dem Server Replay-Noncen sowie JWS. Um die Replay-Nonce zu erhalten, reicht es, einen HEAD-Request an die von "Pebble" für diesen Zweck bereitgestellte Schnittstelle zu senden. Der Aufbau dieser Methode ist unabhängig von der Art der Challenge und gilt somit für EK genauso wie für DNS und HTTP und wird im folgenden Listing beschrieben.

```
func (n dummyNonceSource) Nonce() (string, error) {
    if globNonce != "" {
        return globNonce, nil
    }
    tr := &http.Transport{
        TLSClientConfig: &tls.Config{InsecureSkipVerify: true},
    }
    client := &http.Client{Transport: tr}

    res, err :=
       ↪ client.Head("https://192.168.1.2:14000/nonce-plz")
    if err != nil {
        panic(err)
    }
    ua := res.Header.Get("Replay-Nonce")
    return ua, nil
}
```

Listing 4.3: Clientseitige Beschaffung der Noncen, encryption.go

In der hier beschriebenen Methode wird zuallererst geprüft, ob bereits eine Nonce, hier "globNonce" genannt, vorhanden ist. Falls nicht, wird ein Request ausgesendet und der Response-Header für die Replay-Nonce ausgelesen. Der erste Schritt ist deshalb wichtig, da im folgenden Ablauf jede Antwort des Servers eine neue Nonce übersendet, die den Wert von "globNonce" überschreibt. So muss der Client nicht nach jeder Kommunikation erst eine neue Nonce vom Server erfragen.

Zum Signieren der Nachrichten wird clientseitig ein Schlüsselpaar generiert. Wie im theoretischen Teil bereits besprochen, ist es sinnvoll, das Schlüsselpaar vom TPM-Chip generieren zu lassen. Beispielcode aus der Methode für den Kontoerstellungs-Request:

```
var signerOpts = jose.SignerOptions{NonceSource:
   ↪ dummyNonceSource{}}
signerOpts.WithHeader("jwk", jose.JSONWebKey{Key:
   ↪ globPrivateKey.
    Public()})
signerOpts.WithHeader("url", signMeUpURL)
signer, err := jose.NewSigner(jose.SigningKey{Algorithm:
   ↪ jose.RS256,
    Key: globPrivateKey}, &signerOpts)
if err != nil {
  panic(err)
}
```

Listing 4.4: Anfrage zum Erstellen eines neuen Kontos, encryption.go

Nur in diesem Request, das im Listing "Anfrage zum Erstellen eines neuen Kontos" beschrieben wird, wird der öffentliche Schlüssel verschickt. Im späteren Verlauf, sobald das Konto erstellt wurde, wird anstelle von "jwk" die "kid", die vom Server mitgeteilt wurde, verwendet. Durch diesen Request ist der Server nicht nur in der Lage, ein neues Konto zu erstellen, sondern kann auch den entsprechenden öffentlichen Schlüssel diesem Konto zuordnen. Da alle Nachrichten als POST-Request verschickt werden, kann durch die Signatur geprüft werden, ob es sich dabei um den gleichen Absender handelt wie bei vorangegangen Nachrichten.

4.2.2 Erweiterung des ACME-Protokolls um die neue Challenge

Die neue Challenge umfasst wie die anderen Challenges auch drei Schritte. Erst muss eine Order beim Server aufgegeben werden, dann muss die Challenge aktiviert und anschließend erfüllt werden. Dieser Prozess, zusammen mit dem Anfragen und Verwenden des Zertifikates, soll in diesem Unterkapitel beschrieben werden. Die Idee hinter jeder Änderung bezieht sich dabei auf den im theoretischen Teil beschriebenen Ablauf.

4.2.2.1 Order platzieren

Nachdem der Client registriert ist, kann dieser ein neues Zertifikat anfragen. Dazu soll die neu definierte EK-Challenge verwendet werden, wofür der Client eine Verbindung mit dem Chip herstellt und so den öffentlichen Schlüssel des EK ausliest und sich einen AK erstellen lässt. Für beide Funktionalitäten kann auf das Projekt “go-attestation” [4] von “Google” zurückgegriffen werden. Der AK besitzt sogenannte “Attestation Parameter”, die später vom Server verwendet werden können. Diese Informationen werden nun an den Server gesendet. Dazu wird der Wert des “identifier” mit dem “type”:“ek” und “value”:“[ek+ak]” gesetzt. Der Body des Get-Order-Requests, der an den Server gesendet wird, sieht so aus:

```
{
    "status": "pending",
    "expires": "2021-08-24T13:32:22Z",
    "identifiers": [
        {
            "type": "ek",
            "value": "{\KeyEncoding\":2,\TPMVersion\":2,
                [...]
                -----END RSA PUBLIC KEY-----\n"
        }
    ],
    "finalize":
       ↪ "https://192.168.1.8:14000/finalize-order/dn3VRQ7wgq
    oe7Gu6xnAZQCdRRAbeyKWqoqnloxYsuWs",
    "notBefore": "2021-08-01T00:04:00+04:00",
    "notAfter": "2021-08-08T00:04:00+04:00",
    "authorizations": [
        "https://192.168.1.8:14000/authZ/nQxyaQV942uQoT1AFeM
        FRaFpGQUZ7MiZB_HmK-xAJK4"
    ]
}
```

Listing 4.5: Server-Antwort auf Challenge-Anfrage

Um mit dem neuen "identifier" etwas anfangen zu können, muss der Server um den Identifier sowie die neue "ek-01"-Challenge erweitert werden. Das folgende Codebeispiel soll dabei beispielhaft für die gesamte Implementierung der neuen "identifier" gelten:

```
const (
    [...]

    IdentifierDNS = "dns"
    IdentifierIP  = "ip"
    IdentifierEK  = "ek"  // <-

    ChallengeHTTP01    = "http-01"
    ChallengeTLSALPN01 = "tls-alpn-01"
    ChallengeDNS01     = "dns-01"
    ChallengeEK        = "ek-01" // <-

    HTTP01BaseURL = ".well-known/acme-challenge/"

    ACMETLS1Protocol = "acme-tls/1"
)
```

Listing 4.6: Serverseitiger Eintrag der neuen Identifier, common.go

Nun kann bereits die erste Prüfung stattfinden. Ist der EK-Wert dem Server nicht bekannt, stimmt dieser also nicht mit dem hartkodierten Wert überein, so wird dem Client hier schon ein Fehler zurückgegeben und die Kommunikation endet. Die einzige Möglichkeit für den Client, diesen Schritt zu meistern, ist also über den korrekten öffentlichen Schlüssel des EK-Werts zu verfügen.

4.2.2.2 Challenge aktivieren

Der Server erkennt, dass es sich um einen "ek"-Identifier handelt, und die einzige Challenge, die dem Client für diesen Identifier zur Verfügung steht, ist die "ek-01"-Challenge. Als Vorbereitung für diese Challenge benötigt der Server das Geheimnis, im folgenden Codebeispiel als Secret bezeichnet, welches er zur Überprüfung des Clients verwenden kann. Dazu werden die Werte, die der Client im AK übersendet hat, zusammen mit dem Wert des EK verwendet, um das Geheimnis zu erstellen.

```
params := attest.ActivationParameters{
```

```
    TPMVersion: attest.TPMVersion20,
    AK: attest.AttestationParameters{

        Public:            bpublic,
        CreateData:        bcreateData,
        CreateAttestation: bcreateAttestation,
        CreateSignature:   bcreateSignature,
    },
    EK: getEkPublicKey(ek),
}
secret, encryptedCredentials, err := params.Generate()
if err != nil {
    panic(err)
}
```

Listing 4.7: Geheimnis (secret) Erstellung, wfe.go

In dem vorangegangen Codebeispiel wird die Generierung des Geheimnisses auf der Serverseite dargestellt. Das "b" vor "public", "createData", "CreateAttestation" und "CreateSignature", die jeweils aus dem AK extrahiert wurden, gibt dabei an, dass es sich um byte-Werte handelt. Das so erstellte Geheimnis kann der Client mithilfe eines POST-as-GET-Request abfragen. Der Client muss nun nur noch die Challenge mit dem in "go-attestation" beschriebenen Verfahren lösen.

4.2.2.3 Challenge erfüllen

Im Gegensatz zur HTTP- oder DNS-Challenge, in denen der Client den Server nun aktiv werden lassen würde, muss hier der Client selbst das entschlüsselte Geheimnis an den Server senden. Dieser prüft nun, ob der Wert des gelösten Geheimnisses dem erwarteten Wert entspricht, und entscheidet so, ob die Challenge erfolgreich erfüllt wurde oder nicht. Dieser Prozess der Überprüfung, sowie das Aktualisieren des Status der Challenge kann einige Minuten dauern. Deshalb wird hier ein einfacher Polling-Mechanismus verwendet. Dazu wird ein den Status abfragender Request im Zwei-Sekunden-Takt so lange an den Server gesendet, bis dieser den Status der Challenge geändert hat.

4.2.2.4 CSR und Zertifikat

Da der private Schlüssel den Chip nicht verlassen darf, ist es notwendig, dass der CSR über den TPM-Chip generiert wird. Informationen wie der Name oder die Mailadresse sind natürlich selbst ausfüllbar. Ein Request wird nun mitsamt der CSR an den Server

gesendet, wobei der Server überprüft, ob der Wert des im CSR beschriebenen öffentlichen Schlüssels mit dem des AK übereinstimmt. Ist das der Fall, so stellt der Server das Zertifikat aus. Der Client kann es sich nun per POST-as-GET-Request abholen. Das Einzige, was der Client jetzt noch zu erledigen hat, ist das Zertifikat für den Benutzer des Client-Systems auf dem TPM-Chip zur Verfügung zu stellen.

5 Evaluation

Die Evaluation soll zeigen, ob die in dieser Arbeit besprochenen Erweiterungen wie geplant funktionieren und eine sinnvolle Erweiterung des RFC8555 darstellen. Gleichzeitig wird geprüft, ob das Ziel dieser Arbeit, das sichere Verteilen von X.509-Zertifikaten auf Linux-basierten Endbenutzersystemen, erfüllt wurde. Dazu wird zunächst ein Testlauf mit den entsprechenden Ergebnissen dargestellt. Anschließend wird die neue Challenge mit vorherigen Challenges verglichen und auf mögliche Angriffsvektoren, welche durch die in dieser Arbeit beschriebene Vorgehensweise aufgekommen sind, sowie mögliche sinnvolle Erweiterungen eingegangen.

5.1 Testlauf der ACME-Erweiterung

Um die ACME-Erweiterung zu testen, wurde auf dem eigenen Rechner der ACME-Server und auf einem Raspberry PI mit TPM-Chip der Client ausgeführt. Ziel des Ablaufes war es sicherzustellen, dass die im praktischen Teil beschriebenen client- und serverseitigen Erweiterungen wie geplant funktionieren. Dazu gehören Funktionalitäten wie das Anlegen eines Kontos, das Erstellen und Verwenden der neuen Challenge, die Identifizierung des Endbenutzersystems sowie die neue Art, mithilfe des TPM-Chips einen CSR zu erstellen. Im Folgenden wird der Output der clientseitigen Implementierung dargestellt. Der Status 400 wird erhalten, da serverseitig die entsprechenden Informationen noch nicht zur Verfügung stehen. Wie man sehen kann, wird hier die Nachricht "authChallenge: POST-as-GET request to retreive challenge details" mehrfach ausgegeben. Das liegt an der Umsetzung des Polling-Verfahrens, das den Status so lange abfragt, bis dieser "valid" ist.

```
start
newAccount: Account created!

HTTP result status:  201 Created
newCertificate: New Certificate requested!

```

```
HTTP result status:  200 OK
authChallenge: POST-as-GET request to retreive challange details

The payload with secret looks like this: 0x86c600
HTTP result status:  200 OK
authChallengeAnswer: Challenge answer was send!

HTTP result status:  400 Bad Request
authChallenge: POST-as-GET request to retreive challange details

Value ist : 400
HTTP result status:  200 OK
authChallenge: POST-as-GET request to retreive challange details

Value ist : "pending"
HTTP result status:  200 OK
authChallenge: POST-as-GET request to retreive challange details

Value ist : "valid"
HTTP result status:  200 OK
makeCSRRequest: CSR Request send!

HTTP result status:  200 OK
downloadCertificate: Get URL

HTTP result status:  200 OK
POST-as-GET request to retreive Certificate
```

Listing 5.1: Ablauf der neuen Challenge

5.2 Ergebnisse

Der Output beschreibt den Ablauf des ACME-Protokolls, erweitert um die “ek”-Challenge. Am Ende dieser Kommunikation befindet sich im TPM-Chip das Zertifikat, das für einen Benutzer des Client-Geräts zugänglich gespeichert wurde, sowie dessen privater Schlüssel, der für den Nutzer unzugänglich gespeichert wurde. Da als Betriebssystem für den Pi Linux verwendet wurde, ist das Ziel dieser Bachelorarbeit, das Verteilen von X.509-Zertifikaten auf Linux-basierten Endbenutzersystemen, damit erfüllt.

5.3 Vergleich der EK-, DNS- und HTTP-Challenge

Um die Challenge-Typen vergleichen zu können, wird wiederholt, was die jeweiligen Challenges ausmacht, bevor sie auf Gemeinsamkeiten und Unterschiede geprüft werden.

Bei der HTTP- sowie der DNS-Challenge stellt der Server einen Token zur Verfügung, der die jeweiligen Challenges genau definiert. Diesen Token wandelt der Client, zusammen mit seinem Account Key, in einen Autorisierungsschlüssel um. Bei der DNS-Challenge wird zusätzlich noch mit dem SHA-256-Verfahren ein Hashwert gebildet. Anschließend werden die entsprechenden Werte base64-kodiert und als HTTP-Ressource beziehungsweise als "TXT Resource Record" zur Verfügung gestellt. Der Client sendet nun einen Request an den Server, um diesen wissen zu lassen, dass die Ressource nun für ihn zur Verfügung steht. Der Server fragt diese Information ab. Stimmt der Account Key mit dem vom Server generierten Wert überein, wurde die Challenge erfüllt.

Für die EK-Challenge muss zuerst der EK-Wert aus dem TPM-Chip gelesen und ein AK-Wert mithilfe des Chips generiert werden. Diese Werte bilden zusammen mit dem "ek"-Type den Identifier dieser Challenge und werden so dem Server übergeben. Dieser generiert nun aus beiden Werten ein Geheimnis, das der Client anfragen und lösen muss. Ist dies erreicht, übersendet der Client das gelöste Geheimnis, base64-kodiert, zurück an den Server. Wurde das Geheimnis korrekt gelöst, gilt hier die Challenge als erfüllt.

Die wichtigste Gemeinsamkeit ist, dass in jeder der drei Challenge-Arten bewiesen werden muss, dass der Client die Kontrolle über den Wert im "identifier", egal ob EK, DNS oder Website, besitzt. Dadurch, dass nur die Kontrolle validiert wird, ist die Integrität des Systems irrelevant für das ACME-Protokoll. Auch wenn es Möglichkeiten für die EK-Challenge gibt, die Systemintegrität zumindest teilweise zu überprüfen, was in Kapitel 5.5 besprochen wird, gibt es keinerlei Möglichkeiten für den Server sicherzustellen, dass sich nicht irgendeine dritte Partei die Kontrolle über Chip, Website oder DNS verschafft hat.

Einer der größten Unterschiede zwischen den alten Challenges und der neuen ist die Art der Überprüfung. Wo bei der DNS- und HTTP-Challenge der Server selbst aktiv werden muss, um den Autorisierungsschlüssel abzufragen, nimmt er in der EK-Challenge eine rein passive Rolle ein. Der Server muss seinerseits keinerlei Anfragen erstellen, was bei der HTTP-Challenge sogar zu Komplikationen führen kann. Werden beispielsweise mehrere Webserver verwendet, muss sichergestellt werden, dass auf allen der Autorisierungsschlüssel existiert [17]. Einen weiteren Unterschied stellt die Art der Schlüsselgenerierung und -verwendung dar. Dadurch, dass der AK-Wert zusammen mit dem EK-Wert validiert wird, kann sichergestellt werden, dass beide Werte aus dem gleichen TPM-Chip stammen. Auch wenn eine neue Order aufgegeben wird, muss der Client diese Prüfung erneut

antreten, um wiederholt zu bestätigen, dass EK- und AK-Wert aus dem gleichen Chip stammen. Aus diesem AK-Wert wird nun die CSR generiert, sodass der Server, sofern er den AK-Wert zusammen mit dem EK-Wert gespeichert hat, nur durch das Zertifikat genau identifizieren kann, welches Gerät gerade kommuniziert.

5.4 Angriffsvektoren

In diesem Kapitel werden allgemein mögliche sowie EK-Challenge-spezifische Angriffsvektoren besprochen, die entweder durch die neue Challenge mit ihrer Infrastruktur dazugekommen sind oder aus dem generellen Aufbau des ACME-Protokolls entstehen.

Wie im letzten Kapitel bereits besprochen, prüft der ACME-Server nie die Integrität des anfragenden Systems. Gelingt es einem Angreifer, die Kontrolle über den TPM-Chip zu erlangen, kann er sich über das ACME-Protokoll Zertifikate beschaffen. Kritische Informationen wie private Schlüssel aus dem Chip zu extrahieren, ist jedoch nicht möglich.

Der Server kann durch einen "Denial of Service (DoS)"-Angriff lahmgelegt werden. Hierbei wird der Server durch eine übermäßige Anzahl an Anfragen lahmgelegt. So können zwar keine Informationen extrahiert werden, das Ausstellen von Zertifikaten wie auch die Verifikation bereits vorhandener Zertifikate kann dabei jedoch blockiert werden. Da der ACME-Server nicht nur als Website, sondern auch als CA funktioniert, kann, je nach Architektur des Servers, durch einen solchen Angriff großer Schaden entstehen. Wenn ein neuer Kommunikationspartner auftritt, kann jedes Gerät zur Überprüfung des Zertifikats des Gesprächspartners eine Anfrage an die CA stellen, um sicherzustellen, dass das Zertifikat auch wirklich von ihr ausgestellt wurde. Können Zertifikate nicht mehr verifiziert werden, ist es möglich, dass Kommunikation grundsätzlich abgelehnt wird. Hier gibt es verschiedene Optionen, den Server zu schützen [18] [19] [20].

Es existieren noch einige andere Angriffsmöglichkeiten, die nur kurz angesprochen, aber nicht länger behandelt werden sollen, da sie unwahrscheinlich sind oder praktisch keinen Nutzen für den Angreifer bedeuten. So kann clientseitig das Entfernen oder Zerstören des TPM-Chips die Kommunikation lahmlegen, denn ohne Zertifikat mit entsprechendem privaten Schlüssel ist verschlüsselte Kommunikation unmöglich. In diesem Fall muss mindestens der Chip sowie der entsprechende Eintrag in der Serverdatenbank ausgetauscht werden. Wenn ein Angreifer es schafft, sich die Kontrolle über den Server anzueignen, ist er in der Lage, jedwede Kommunikation zu erlauben, sodass effektiv keine Sicherheit mehr gewährleistet wird. Außerdem kann er die Datenbankeinträge löschen, was zu Problemen führen kann, wenn keine Backups gemacht wurden. Im schlimmsten Fall müsste jeder TPM-Chip ausgetauscht werden, da nicht mehr sichergestellt werden kann, ob es

sich um den öffentlichen Schlüssels des EKs des TPMs handelt, der vor oder während des Angriffs in die Datenbank geschrieben wurde.

5.5 Mögliche Erweiterungen

In diesem Kapitel sollen alle alternativen Umsetzungsmöglichkeiten sowie mögliche Erweiterungen besprochen werden. Dabei soll unterschieden werden zwischen clientseitigen Änderungen und serverseitigen Änderungen.

Wie bereits angesprochen, gibt es clientseitig, durch den TPM-Chip die Möglichkeit, die Integrität des Clients beim Boot-Vorgang zu überprüfen. Vereinfacht kann gesagt werden, dass zu Beginn des Boot-Vorgangs geprüft wird, ob das System so ist, wie es sein sollte. Dazu wird ein Startpunkt, ein sogenannter "Core Root of Trust for Measurement" erzeugt. Bei jedem Boot-Vorgang wird nun ein Wert erzeugt, der mit einem Hashverfahren in den Chip gespeichert wird. Weicht dabei ein Wert von den vorherigen ab, kann davon ausgegangen werden, dass das System nicht mehr in seinem originalen Zustand existiert [21]. So kann zwar nicht verhindert werden, dass eine dritte Partei sich die Kontrolle über den Chip aneignet. Sollte jedoch der Fehler gemacht werden und das Gerät zu irgendeinem Zeitpunkt ausgeschaltet werden, so kann dieses nicht wieder neu hochfahren. Durch die aktuelle Implementierung des Clients kann dieser nicht auf mögliche Störungen des Servers reagieren. Eine sinnvolle Erweiterung könnte sein, Fehlerbehandlungen durchzuführen, falls dieser nicht erreicht werden kann oder Anfragen nicht richtig bearbeitet werden.

Serverseitig kann, sobald ein Client ein neues Konto anlegt, der öffentliche Teil seines Account Key in der Datenbank gespeichert werden. Übersendet nun dieser Client in einem New-Order-Request seinen EK- und AK-Wert, können alle drei Werte zusammen in der Datenbank verknüpft werden. Dadurch wird der Server im späteren Verlauf deutlich leichter handzuhaben. Wird beispielsweise eines der Geräte als vermisst gemeldet und sollen alle Zertifikate widerrufen werden, so kann der Server durch die Verknüpfung zwischen den Konto-Daten und dem EK nicht nur genau sagen, welches Client-Konto zu dem verlorenen Chip gehört und Auskunft darüber geben, was dieser in der letzten Zeit für Anfragen gestellt hat. Durch die logs ist es ihm auch möglich, sofort alle zugehörigen Zertifikate zu widerrufen, da diese durch die Verknüpfung mit der EK alle eindeutig diesem Client zugeordnet sind. Auch die Verwendung des EK-Zertifikates statt des in der Arbeit beschriebenen öffentlichen Schlüssels wäre möglich. So kann vor dem Eintragen des Wertes in der Datenbank das Zertifikat geprüft werden. Auch die Gültigkeit dieses Zertifikates kann immer wieder serverseitig überprüft werden, um sicherzustellen, dass der Hersteller nachträglich keine Fehler im Gerät entdeckt hat.

Wie bereits besprochen, wurde für die Umsetzung keine richtige Datenbank verwendet, sondern hartkodiert. Eine Möglichkeit, den Server sinnvoll zu erweitern, wäre das Hinzufügen einer solchen Datenbank mit entsprechender Infrastruktur, sowie das Bereitstellen einer API, die es autorisierten Personen erlaubt, Einträge in die Datenbank zu schreiben. Zugleich kann es auch sinnvoll sein, es Systemadministratoren zu erlauben, Zertifikate zu widerrufen, sollte beispielsweise ein Endbenutzer-System verloren gegangen sein.

Zusätzlich kann der Server auf einem "Docker-Container"[22] laufen gelassen werden. Abgesehen davon, dass der "Container" alle vom Server benötigten Ressourcen zur Verfügung stellt, kann so ein Serverstatus festgelegt werden, auf den immer wieder zurückgesetzt werden kann, falls es zu Komplikationen kommen sollte.

Eine Alternative zu dem im RFC8555 beschriebenen Vorgehen zum neuen Ausstellen von Zertifikaten ist das Verketten von Zertifikaten, auch "Certificate Chaining" genannt[13]. Grob gesagt kann der Server, wenn er dem bereits vorhanden Zertifikat vertraut, falls mit diesem ein neues Zertifikat angefragt wird, transitiv auch dem neuen CSR vertrauen.

6 | Fazit

Abschließend und aufbauend auf der Evaluation, soll in diesem Kapitel die gesamte Arbeit zusammengefasst werden. Zunächst wird erläutert, warum diese Bachelorarbeit geschrieben wurde, ob die Ziele erreicht wurden und was sie von vergleichbaren Arbeiten abhebt. Daran anschließend werden weitere Schritte besprochen, die kurz die Verwendung des Protokolls in einer praktischen Umgebung beschreiben, und dargestellt, wie das RFC8555 durch diese Arbeit sinnvoll erweitert werden kann.

6.1 Zusammenfassung

In dieser Arbeit wurde die Frage behandelt, wie eine sichere Verteilung von X.509-Zertifikaten auf Linux-basierten Endbenutzersystemen aussehen kann. Dabei wurde der Schwerpunkt nicht nur auf die Verteilung von X.509-Zertifikaten, sondern auch auf deren Speicherung und Verwendung gelegt. Durch diese Arbeit soll es möglich gemacht werden, die Funktionalität, die für Windows-Systeme bereits zur Verfügung steht, auch Linux-basierten Systemen zu ermöglichen. Dabei wurde sich dafür entschieden, auf bereits existierende Projekte aufzubauen. Für das Erstellen und Verwalten von Zertifikaten sollte auf ACME zurückgegriffen werden und für die Prüfung von Endgeräten auf "go-attestation" von "Google". Durch die Verbindung dieser Projekte und unter der Verwendung eines Chips, der eine eindeutige Identifikation erlaubt, war es möglich, für einen Pi, auf dem ein Linux-System lief, einen entsprechenden Client zu erstellen. Um dieses Ziel der Forschungsfrage entsprechend umzusetzen, sind zwei Akteure vonnöten. Der erste stellt das Linux-basierte Endbenutzersystem dar, das ein X.509-Zertifikat erhält. Das andere ist die Instanz, die ein solches Zertifikat zur Verfügung stellt. Sowohl ACME als auch "go-attestation" arbeiten dabei nach einem gleichen Client-Server-Modell, wodurch es möglich war, diese Projekte sinnvoll miteinander zu verknüpfen, um diese Bachelorarbeit zu verfassen. Der erste Schritt bestand dabei darin, die ACME-Kommunikation zu analysieren und um eine neue Challenge zu erweitern. Diese Challenge stellt einen Sicherungsmechanismus für die Verteilung der Zertifikate dar. Denn dadurch ist es serverseitig möglich, das Endbenutzersystem eindeutig zu identifizieren. Ein ähnliches Verfahren wie

das unter der Verwendung des TPM-Chips wurde für ACME bereits entwickelt. Dabei handelt es sich um zwei alternative Verfahren, die IP-Adressen oder Telefonnummern für die Identifikation des Endbenutzersystems verwenden. Auch wenn beide in der Lage sind, genau wie das in dieser Arbeit beschriebene Verfahren, Zertifikate auszustellen und das Endbenutzersystem zu identifizieren, gibt es gute Gründe, das neue Verfahren zu implementieren. Zum einen dient die Verwendung von Telefonnummern zur Prüfung weniger der Prüfung des Endbenutzersystems und mehr der des Nutzers des Systems. Zum anderen sind IP Adressen einem ständigen Wechsel ausgesetzt, was es dem Server schwierig macht, das genaue Endbenutzersystem zu definieren. Durch diese Arbeit ist es nun – wie in der Evaluation bewiesen werden konnte – möglich, ein Endbenutzersystem eindeutig zu identifizieren.

6.2 Future Work

Der nächste Schritt dieser Arbeit besteht darin, für das RFC8555 eine Erweiterung zu schreiben. Dieses Dokument kann nach einigen Korrekturen und Überarbeitungen eine Erweiterung für das klassische ACME-Protokoll darstellen, wie auch die IP und Telefonnummern Erweiterungen für ACME sind.

Ist diese Erweiterung ein fester Bestandteil des ACME-Protokolls, kann darauf aufgebaut werden. Eine beispielhafte Verwendung dieses Verfahrens könnte sein: Möchte eine größere Firma sicherstellen, dass alle Geräte, mit denen kommuniziert wird, über ein eigenes Zertifikat verfügen, so kann die Kommunikation auf der Prämisse aufbauen, dass sich, nachdem ein Zertifikat angefragt wurde, im TPM-Chip ein solches befindet. Diese Zertifikate können verwendet werden, damit alle Kommunikationspartner stets wissen, mit wem sie gerade Informationen austauschen. Der ACME-Server fungiert dann auch gleichzeitig als Anfragemöglichkeit für jeden Gesprächspartner, der das Zertifikat des jeweils anderen Gesprächspartners prüfen möchte. Weiterführend wäre es auch sinnvoll, dieses Projekt statt in die abgeschwächte Version eines ACME-Servers (Pebble) in einen vollwertigen ACME-Server (zum Beispiel Boulder) zu implementieren.

Literatur

[1] D. McCarney R. Barnes J. Hoffman-Andrews. 2019. Automatic Certificate Management Environment (ACME). Abgerufen 25. August 2021 von https://tools.ietf.org/html/rfc8555

[2] R. B. Shoemaker. 2020. Automated Certificate Management Environment (ACME) IP Identifier Validation Extension. Abgerufen 13. August 2021 von https://datatracker.ietf.org/doc/html/rfc8738

[3] 2019. Making Device Identity Trustworthy. Abgerufen 25. August 2021 von https://static.sched.com/hosted_files/osseu19/ec/Device%20Identity.pdf

[4] R. B. Shoemaker. Go-Attestation v0.3.2. Abgerufen 18. August 2021 von https://github.com/google/go-attestation

[5] ISO/IEC 11889-1:2009. Abgerufen 7. August 2021 von https://www.iso.org/standard/50970.html

[6] Pierpaolo Degano; Sandro Etalle; Joshua Guttman. 2016. *Formal Aspects of Security and Trust*. Springer.

[7] Christof Windeck. 2021. Trusted Platform Module 2.0 in Windows 11. Abgerufen 7. August 2021 von https://www.heise.de/ratgeber/Trusted-Platform-Module-2-0-in-Windows-11-6135986.html

[8] 2019. Trusted Platform Module Library Part 1: Architecture. Abgerufen 22. August 2021 von https://trustedcomputinggroup.org/wp-content/uploads/TCG_TPM2_r1p59_Part1_Architecture_pub.pdf

[9] Trusted Platform Module (TPM) Summary. Abgerufen 22. August 2021 von https://trustedcomputinggroup.org/resource/trusted-platform-module-tpm-summary/

[10] Will Arthur; David Challener; Kenneth Goldman. 2015. *A Practical Guide to TPM 2.0*. Apress open.

[11] R. Fielding; J. Reschke. 2014. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. Abgerufen 9. August 2021 von https://datatracker.ietf.org/doc/html/rfc7231#page-24

[12] R. B. Shoemaker. 2020. Automated Certificate Management Environment (ACME) TLS Application-Layer Protocol Negotiation (ALPN) Challenge Extension. Abgerufen 13. August 2021 von https://datatracker.ietf.org/doc/html/rfc8737

[13] 2021. Chain of trust. Abgerufen 24. August 2021 von https://en.wikipedia.org/wiki/Chain_of_trust

[14] Pebble. Abgerufen 24. August 2021 von https://github.com/letsencrypt/pebble

[15] pebble (forked from letsencrypt/pebble). Abgerufen 10. September 2021 von https://github.com/RRToast/pebble

[16] ACMEclient. Abgerufen 10. September 2021 von https://github.com/RRToast/ACMEclient

[17] Challenge Typen. Abgerufen 22. August 2021 von https://letsencrypt.org/de/docs/challenge-types/

[18] 2018. Abwehr von DDoS-Angriffen. Abgerufen 23. August 2021 von https://www.allianz-fuer-cybersicherheit.de/SharedDocs/Downloads/Webs/ACS/DE/BSI-CS/BSI-CS_002.pdf;jsessionid=67F80FFF524489B8981810312FE7F701.internet082?__blob=publicationFile&v=1

[19] 2018. Anti-DDoS-Maßnahmen. Abgerufen 23. August 2021 von https://www.allianz-fuer-cybersicherheit.de/SharedDocs/Downloads/Webs/ACS/DE/BSI-CS/BSI-CS_090.pdf;jsessionid=67F80FFF524489B8981810312FE7F701.internet082?__blob=publicationFile&v=1

[20] 2018. Prävention von DDoS-Angriffen. Abgerufen 23. August 2021 von https://www.allianz-fuer-cybersicherheit.de/SharedDocs/Downloads/Webs/ACS/DE/BSI-CS/BSI-CS_025.pdf;jsessionid=67F80FFF524489B8981810312FE7F701.internet082?__blob=publicationFile&v=1

[21] 2019. Trusted Platform Module Library Part 1: Architecture. 29–30. Abgerufen 23. August 2021 von https://trustedcomputinggroup.org/wp-content/uploads/TCG_TPM2_r1p59_Part1_Architecture_pub.pdf

[22] Use containers to Build, Share and Run your applications. Abgerufen 24. August 2021 von https://www.docker.com/resources/what-container