% Abschlussarbeit

% Autor_in

% Prüfer_in

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

01.03.2018

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, 01.03.2018

. . . . . .

Unterschrift

Abstract

In dieser Arbeit wird die Fragestellung behandelt, wie es möglich ist für Linux-basierten Endbenutzersysteme Zertifikate zu verteilen. Zu diesem Zweck wurde mithilfe des TPM-Moduls eine neue ACME Challenge entwickelt. Mit der neuen sogennanten "EK" Challenge ist es dem Endbenutzersystem möglich sich gegenüber eines ACME Servers zu verifizieren. Um das Verfahren umzusetzen wurde, statt einen neuen ACME Server zu schrieben auf ein Projekt von letsencrypt das sich pebble nennt zurückgegriffen. Dabei handelt es sich um eine vereinfachte Version eines Servers welcher in dieser Arbeit so umgebaut wurde, dass es einem ebenfalls in dieser Arbeit geschrieben Client möglich ist die neue Challenge zu verwenden. Das Verfahren funktioniert und kann verwendet werden um das ACME Protokoll zu erweitern. Im Rahmen dieser Arbeit werden auch weiterführende Sicherheitsmaßnahmen besprochen, welcher man sich bei der Umsetzung dieses Projektes bedienen kann.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

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 und deren Verlust kann schwerwiegende Folgen haben. 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 z.B. private Daten abzufragen oder Industriespionage zu betreiben. Dabei wird unterschieden zwischen der Prüfung von Nutzern und Endgeräten. Eine Kommunikation kann von Anfang an abgelehnt werden, wenn sichergestellt werden kann, 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 unbrauchbar wird. Gleichzeitig muss ein Grad an Sicherheit vorherrschen, der es so schwer wie möglich macht das Zertifikat zu missbrauchen. 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 fängt vor dem Ausstellen des Zertifikates an und hört erst auf, 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, dabei soll nicht das Windows Projekt kopiert, sonder mithilfe verschiedener Verfahren und Prinzipien ein neues Verfahren entwickelt werden. Damit

das Zertifikat, welches durch das neuen 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 TLS verwendet. Zusammenfassend kann also gesagt werden, dass es das Ziel dieser Bachelorarbeit, die sichere Verteilung von X.509 Zertifikaten unter Linux-basierten Endbenutzersystemen zu ist.

1.3 Verwandte Arbeiten

Aus der Forschungsfrage können drei Fragen abgeleitet werden: Einmal, wie kann das Endbenutzersystemen eindeutig identifiziert werden? Zweiten, wie können die entsprechenden X.509-Zertifikate erstellt und verwaltet werden. Drittens, wie können die Zertifikate sicher von der Zertifizierungsstelle zum 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) veröffentlicht[1]. Diese behandlet eine automatisierte Prüfung, sowie die automatisiert Austellung von Zertifikaten für die entsprechendne Systeme. Dieses Paper bildet eine wunderbare Grundlage für diese Arbeit, ist jedoch alleine nicht ausreichend, da Gerätvaliderung nicht möglich ist. Es gibt Erweiterungen dieses Protokolls und eine davon, die Prüfung der IP Adresse[2] soll hier noch kurz erwähnt werden. Dabei handelt es sich zwar um eine Eigenschafft eines Systems, jedoch reicht es nicht aus um dieses eindeutig zu identifiziert. Auch Google hat sich dem Problem der eindeutigen Identifikation von Geräten gewidmet. So beschreiben sie in einem Vortrag[3], eine Vorgehensweise die es erlaubt unter Verwendung von TPM-Chips Geräte eindeutig zu identifiziert. Diese Vorgehensweise, welche im Projekt "go-attestation"[4] beschrieben wurde bildet den zweiten Grundstein dieser Arbeit. Auf diesen Aufbauend soll ein eigenes neues Verfahren entwickelt werden, welche zusätzliche Sicherheitsmechanismen für Endsystembenutzer, darstellt.

1.4 Übersicht über die Bachelorarbeit

Um dieses Ziel 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. Zusätzlcih wird hier auch die Verwendung verschiedener Schutzmaßnahmen von Gerät und Nutzer eingegangen. Im anschließenden 4. Kapitel wird anhand von

praktischen Beispielen erklärt, wie so ein Verfahren konkret umgesetzt werden kann. Abschließend wird die Arbeit genau analysiert und unter anderem auf Funktionalität sowie Schwachstellen überprüft. Hier wird auch ein kleiner Ausblick gegeben was anders hätte umgesetzt werden können und welche Vor- und Nachteile das mit sich bringen würde. Abschließend wird in einem Fazit die Arbeit noch einmal zusammengefasst und mögliche nächste Schritte besprochen.

2 | Grundlagen

Diese 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 Zertifikate dar. Zuerst werden grundlegende Funktionalitäten des TPM-Moduls besprochen, bevor der Ablauf sowie die wichtigsten Konzepte des ACME Protokolls dargestellt werden.

2.1 TPM

Das "Trusted Platform Module" (TPM) ist 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 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 Betriebsystem Windows 11 TPM2.0 Chips als Anforderung zu stellen, wahrscheinlich weiter steigen wird[7]. Die Idee des TPM-Moduls ist es, die Limitationen und Probleme, die Sicherheitssoftware und standardmäßige Hardware mit sich bringen, zu beheben. Darunter fallen unsicherer Speicher, volatiler Speicher sowie unsichere kryptographische Hardware. Der Aufbau des TPM ist etwas komplexer und wird in dem ofiziellen Dokument der Trusted Computing Group auf etwas über 300 Seiten zusammengefasst[8]. Durch eine eigene API, welche das Modul zur Verfügung stellt, ist es möglich mit diesem zu kommunizieren. Gleichzeitig besitzt das Modul zwei, für diese Arbeit sehr relevante Komponenten. Einmal einen geschützten persistenen Speicher, welcher es erlaubt wichtige Informationen sicher abzuspeichern[6]. Zum anderen eine kryptographische Einheit welche Schlüsselpaare erstellen kann [9]. Dabei wird unter anderem sichergestellt, dass die kryptographischen Verfahren auf Hadware 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 persistenent 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 welches vom Hersteller signiert wurde[10]. An diesem Zertifikat in Verbindung mit dem entsprechenden privaten Schlüssel kann der TPM-Chip eindeutig identifiziert werden.

2.2 ACME

ACME steht für "Automatic Certificate Canagement Enviroment". Diese besteht aus zwei Teilen: erstens einer Intsanz welche Zertifikate verwalten kann, eine sogenannten "Certificate Authority" (CA) welche Zertifikate erstellen und wiederrufen kann und zweitens einer automatisierten Schnittstelle welche 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, welches ein Zertifikat benötigt, ein ACME Client ausgeführt, welcher eine Anfrage an den entsprechenden ACME Server stellt. Dieser prüft dann mit sogenannten Challenges, dass der Client tatsächlich ist der ist, für den er sich ausgibt. Ist der Client in der Lage die Challenge zu erfüllen kann dieser 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 einen Konto erstellen muss, welches 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 RFC-8555 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 Directory und Übersicht über alle URLs die benötigt werden, damit

der Client mit dem Server kommunizieren kann. Darunter finden sich unter anderem URLs um ein Zertifikat zu widerrufen, eine Replay-Nonce zu erhalten und einen Konto zu erstellen. Jede Kommunikation, mit Ausnahme des GET-Requests zum Erhalten des Directorys sowie dem Erhalten der Replay-Nonce, ist verschlüsselt und muss mit einer Replay-Nonce abgesichert werden. Dabei übersendet der Server bei jeder Anfrage des Client auch eine Replay-Nonce im Header des Responses mit, sodass diese nicht jedes Mal neu angefragt werden muss. Jede Kommunikation, ausgenommen der 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 JWS gesichert 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

Folgend 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 der grundsätzliche Ablauf beschrieben werden, in dem der Client noch kein Konto besitzt, also Client und Server noch keinerlei Kontakt miteinander hatten. Der Einfachheit halber wird die Kommunikation aus Sicht des Clients dargestellt und die internen Abläufe des ACME Servers, da sie von Server zu Server unterschiedlich sein können, außer Acht gelassen. Zu allererst 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 Die erste Nonce

Die Nonce wird im weiteren Ablauf des Protokolls 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 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 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 2.2.2.1: "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, dass der Client mit den Nutzungsbedingungen einverstanden ist, an den Server. Optional könnte 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 JWK mit dem entsprechendem öffentlichen Schlüssel, der zum Signieren verwendet wurde, an den Server gesendet. Dieses Listing beschreibt dabei den Inahlt 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 2.2.2.2: "7.3 Account Management", RFC8555

Die Antwort des Servers wird musterhaft im nächsten Listing dargestellt. Dabei übersendet der Server neben den "contact" Informationen, welche 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 2.2.2.2: RFC 8555, "7.3 Account Management"

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 sogennanten "type" sondern auch gegen welchen Wert "value" die Validierung stattfinden soll. 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 genau ein Link erstellt, im sogenannten "Authorizations" Array. 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 2.2.2.3: RFC 8555, "7.4 Applying for Certificate Issuance"

2.2.2.4 Challenge aktivieren

Für den Client gibt es mehrere Methoden, mit denen er beim Server validieren kann, dass er tatsächlich ist, wer er zu sein vorgibt. Im RFC8555-Dokument werden dabei zwei näher erläutert, die auch hier ausführlicher besprochen werden sollen, nämlich die HTTP und die DNS Challenge. Weiterführend gibt es Erweiterungen für das Dokument, wie eine Challenge welche die Kontrolle über eine IP Adresse prüft [2], oder die Domain über TLS prüfen [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, an die 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 den type "http-01" oder "dns-01". Beide haben eine eigene URL, sowie einen gemeinsamen Token. Der Token ist dabei

ein zufälliger base64 Wert, der die Challenge eindeutig identifiziert.

DNS Challenge: Der Client kann aus diesem Token, in Verbindung mit seinem eigenen Konto Schlüssel, einen "Authorization Key" erstellen. Dieser Schlüssel wird anschließend mit dem SHA-256 Verfahren verschlüsselt (hashed). Dieser Wert wird nun base64 encoded und in einem TXT Resource Record im DNS gespeichert. Dieses Dokument wird dabei unter der im Identifier Value angegebenen Domain unter dem Prefix "_acme-challenge" abgespeichert. Für "www.example.org" wird der Wert also unter "_acme-challenge.www.example.org" abgespeichert [1]. Der Client sendet nun einen POST-as-GET Request zum Server, um diesen zu informieren, dass dieser die Information anfragen kann.

HTTP Challenge: Die http-01 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

Nun sendet der Client eine POST-as-GET Request an den Server, um ihn wissen zu lassen, dass er nun mit der Validierung beginnen kann. Um zu validieren, dass die Challenge erfüllt wurde, erstellt der Server den selben 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. Ist das gelungen, gilt die Challenge als bestanden.

2.2.2.6 Zertifikatsmanagement

Ist der Status für diese Order als valide gesetzt, wurden also alle Challenges erfüllt, so kann der Client sein Zertifikat anfragen. Dazu sendet er dem Server eine Certificate Signing Request (CSR) an die finalize URL. Anhand dieser CSR erstellt nun der Server das Zertifikat und übersendet dem Client in der Antwort unter anderem die certificate URL, also den Ort, an dem das Zertifikat zur Verfügung steht, mit.

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.

2.2.2.7 Weitere mögliche Schritte

Damit ist die Kommunikation abgeschlossen. 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, welche 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 identifiziert zu können zu erreichen. Auch wie der Vorgang server- und clientseitig sicherer gestalltet 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 Konto Schlüssel 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 Endbenutzersystemen eindeutig identifiziert können.

3.1.1 Vorbereitung

Im Gegensatz zu den beiden oben besprochenen Challenge-Arten soll die neue Challenge die Kontrolle über das Gerät, für welches 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 welches 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 beschrieben 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 dem 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, dass 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 enstprechenden 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 400 Fehler zurückgegeben, kommt der EK Value 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, welcher 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, welches 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

3.2.1 Clientseitig

Der Client verfügt also 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 dem entsprechende Maßnahme ist, dass der rivate 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, sowie der Bearbeitung der neuen EK-Challenge erweitert. Durch ersteres soll es einem Systemadministrator einfach gemacht 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 diesem 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 soll der Aufbau des Systems und allem dazugehörigem besprochen werden. Im zweiten geht es um die Implementierung des Verfahrens welche Anhand von ausgwählten Codebeispielen erklärt werden soll.

4.1 Architektur

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

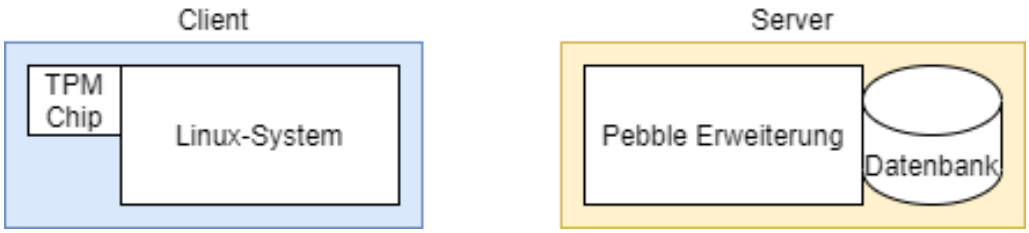

Abbildung 4.1: Vereinfachte Darstellung des Client Server Aufbaus

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 so blockiert. Ein einfacher Test in Linux sieht folgendermaßen aus:

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

Listing 4.1.1: Schnellcheck für TPM erreichbarkeit Antwortet der TPM-Chip mit "resource is busy" blockiert ein anderer Prozess. 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 hartcodiert wie im folgenden Listing zu sehen ist. Wichtig ist nur, dass die Werte persistent gespeichert sind und der Server diese abfragen kann.

```
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 4.1.2: EK öffentlicher Schlüssel, wfe.go

4.2 Implementierung

Das Projekt wurde sowohl server- wie auch clientseitig in der Programmiersprache "GO" geschrieben. Als Server wurde mein eigener 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 Bilder sollen dabei als visuelle Stütze dienen und sind nicht ausreichend um das Projekt nachzubauen.

4.2.1 Implementierung des regulären ACME Ablaufs

Wie bereits in den Grundlagen beschrieben, bedient sich die Kommunikation zwischen dem Client und 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 so 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 4.2.1: clientseitige Noncen beschaffung, encryption.go In der hier Beschriebenen Methode wird zuallererst geprüft ob bereits einen Nonce, hier globNonce genannt, vor-

handen 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, welche 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 4.2.1: Anfrage zum erstellen eines neuen Kontos, encryption.go Nur in diesem Request welches im Listing "Anfrage zum erstellen eines neuen Kontos" beschrieben wird, wird der öffentlichen 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, eine 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

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/dn3VRQ7wgqoe7Gu6xnAZQCd
    "notBefore": "2021-08-01T00:04:00+04:00",
    "notAfter": "2021-08-08T00:04:00+04:00",
    "authorizations": [
        "https://192.168.1.8:14000/authZ/nQxyaQV942uQoT1AFeMFRaFpGQUZ7MiZB_HmK-xA
    ]
}
```

Listing 4.2.2.1: Server Antwort auf Challenge Anfrage

Abbildung 4.2: getOrder Body

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 gesammte 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 4.2.2.1: Serverseitiger Eintrag der neuen Identifier

Nun kann bereits die erste Prüfung stattfinden. Ist der EK Wert dem Server nicht bekannt, stimmt dieser also nicht mit dem hartcodierten 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, welche 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 4.2.2.2: Geheimnis (secret) Erstellung, wfe.go In dem vorran gegangen Codebeispiel wird die generierung des Geheimnisses auf der Serverseite dargesetllt. 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 Statuses 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, das 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 GET-as-POST 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

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 in 4. beschrieben client- und serverseitigen Erweiterung wie geplant funktionieren. Darunter gehören Funktionalitäten, wie das Anlegen eines Kontos, das Erstellen und verwenden der neuen Challenge, die Identifizierung des Endbenutzersystems, sowie die neue Art CSR zu erstellen. Im folgenden wird der Output der Clientseitigen implementiert dargesetllt. Der Status 400 wird erhalten, da Serverseitig die entsprechenden Informationen noch nicht zur verfügung stehen. Wie man sehen kann wird hier die Nachricht "authChallenge: GET-as-POST request to retreive challange details" mehrfach ausgegeben. Das liegt an der Umsetzung des "Polling" Verfahrens, welches den Status so lange abfrägt, bis dieser "valid" ist.

```
start
newAccount: Account created!

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

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

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

Value ist : "pending"
HTTP result status:  200 OK
authChallenge: GET-as-POST 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
GET as POST 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, für einen Benutzer des Client geräts zugänglich gespeichert, sowie dessen privater Schlüssel, für den Nutzer unzugänglich gespeichert. Da als Betriebsystem 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 mit der DNS und HTTP Challenge

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

HTTP und DNS Challenge: In beiden Challenges stellt der Server einen Token zur Verfügung, der die jeweiligen Challenges genau definiert. Diesen Token wandelt der Client, zusammen mit seinem Konto Schlüssel, in einen Authorisierungsschlüssel um. Bei der DNS Challenge wird zusätzlich noch mit dem SHA-256 Verfahren ein Hashwert gebildet. Anschließend werden die entsprechenden Werte base64 codiert 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 Konto Schlüssel mit dem vom Server generierten Wert überein, wurde die Challenge erfüllt.

EK-Challenge: 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" Typ den Identifier dieser Challenge und werden so dem Server übergeben. Dieser generiert nun aus beiden Werten ein Geheimnis, welches der Client anfragen und lösen muss. Ist das geschafft, übersendet der Client das gelöste Geheimnis, base64-codiert, zurück an den Server. Wurde das Geheimnis korrekt gelöst gilt hier die Challenge als erfüllt.

Gemeinsamkeiten: Auffällig ist, dass alle drei Challenge-Arten, nur Kontrolle über etwas beweisen: In jeder muss bewiesen werden, 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 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.

Unterschiede: 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 Authorisierungsschlüssel abfragen, so 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, das auf allen der Authorisierungsschlüssel existiert [15]. 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, insofern 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 sollen allgemein mögliche, sowie EK-Challenge-spezifische Angriffsvektoren besprochen werden, welche entweder durch die neue Challenge mit all ihrer Infrastruktur dazu gekommen 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. Schafft es ein Angreifer, die Kontrolle über den TPM-Chip zu erlangen, kann er sich über ACME 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, aber 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 angerichtet werden. 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, kann es passieren, dass Kommunikation grundsätzlich abgelehnt wird. -> Hier gibt es verschiedene Möglichkeiten den Server zu schützen [16] [17] [18].

Es gibt 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, auch wenn sie problematisch für den Client sein können. So kann clientseitig das Entfernen oder Zerstören des TPM-Chips die Kommunikation lahm legen, denn ohne Zertifikat ohne entsprechenden privaten Schlüssel ist 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 und so effektiv keine Sicherheit mehr zu gewährleisten, außerdem kann er die Datenbank Einträge löschen, was wenn keine Backups gemacht werden, zu Problemen führen kann. Im schlimmsten Fall müsste jeder TPM-Chip ausgetauscht werden, da nicht mehr sichergestellt werden kann ob es sich um einen Chip handelt der vor oder während dem Angriff 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.

Clientseitig: Wie bereits angesprochen gibt es durch den TPM-Chip die Möglichkeit, die Integrität des Clients beim Bootvorgang zu überprüfen. Vereinfacht kann gesagt werden, dass zu Beginn des Bootvorgangs 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 Bootvorgang 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 [19]. 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 möglich Störungen des Servers reagieren. Eine sinnvolle Erweiterung könnte sein Fehlerbehandlungen durchzuführen, falls dieser nicht erreicht werden kann oder Anfragen nicht richtig bearbeitet werden.

Serverseitig: Sobald ein Client einen neues Konto anlegt, kann sein kontogebundener öffentlicher Schlüssel 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 es sollen alle Zertifikate widerrufen werden, so kann der Server durch die Verknüpfung zwischen den Konto-Daten und dem EK nicht nur genau sagen welcher 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 beschrieben öffentlichen Schlüssel 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 sicher zu stellen dass der Herrsteller nachträglich keine Fehler im Gerät entdeckt hat.

Wie bereits besprochen wurde für die Umsetzung keine richtige Datenbank verwendet. Eine Möglichkeit, den Server sinnvoll zu erweitern, wäre das Hinzufügen einer solchen

Datenbank mit entsprechender Infrastruktur, sowie dem Bereitstellen einer API, die es autorisierten Personen erlaubt, Einträge in die Datenbank zu schreiben. Im gleichen Zug 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[20] 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 beschrieben Vorgehen zum neuen Ausstellen von Zertifikaten ist das Verketten von Zertifikaten, auch Certificate Chaining genannt[13]. Grob gesagt, wenn der Server dem bereits vorhanden Zertifikat vertraut, so kann er, falls mit diesem Zertifikat ein Neues angefragt wird, transitiv auch der neuen CSR vertrauen.

6 Fazit

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 bei dieser Arbeit nicht nur ein Schwehrpunkt auf die Verteilung von X.509-Zertifikaten sondern auch auf deren Speicherung und Verwendung gelegt. Mit dieser Arbeit soll es möglich gemacht werden die Funktionalität, welche für Windows Systeme bereits zur Verfügung steht, auch Linux-basierte 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 zu erstellen. Um dieses Ziel, der Forschungsfrage entsprechend umzusetzen sind zwei Akteure von nöten. Der erste stellt das Linux-basierte Endbenutzersystem dar, welches ein X.509 Zertifikat erhält, das andere ist die Instanz, welche ein solches Zertifikat zur Verfügung stellt. Sowohl ACME als auch go-attestation arbeiten dabei nach einem gleichen Client Server Model und so war es möglich diese Projekte Sinvoll mit einander 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 da. Denn dadurch ist es Serverseitig möglich das Endbenutzersystem eindeutig zu identifizieren. Ein ähnliches Verfahren wie dass unter der Verwendung des TPM-Chips wurde für ACME bereits entwickelt. Dabei handelt es sich um zwei alternative Verfahren welche 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 auszugstellen sowie das Endbenutzersystem zu identifizieren, gab es gute Gründe das neue Verfahren zu implementieren. Einmal dient die Verwendung von Telefonnummern zur Prüfung weniger der Prüfung des Endbenutzersystems und mehr der des Nutzers des Systems. Zweitens sind IP Adressen

einem ständigen Wechsel ausgesetzt, was es dem Server schwierig macht das genaue Endbenutzersystem zu definieren.

6.2 Future Work

Der nächste Schritt dieser Arbeit besteht darin für das RFC8555 eine Erweiterung zu schreiben. Dieses draft würde, nach einigen korrekturen und Versionen eine Erweiterung für das klassiche ACME Protokoll darstellen, wie auch die IP und Telefonnummern Erweiterungen darstellen.

Ist das Draft ein fester bestandteil des ACME Protokolls kann darauf aufgebaut werden. Ein beispielhafte Verwendung dieses Verfahrens könnten sein: Möchte eine größere Firma sicherstellen, dass alle Geräte mit denen Kommuniziert werden über ein eigenes Zertifikat verfügen, so kann die Kommunikation auf der Premisse aufbauen, dass sich nachdem ein Zertifikat angefragt wurde, im TPM-Chip ein solches befindet. Diese Zertifikate können verwendet werden, damit alle Kommunikationspartner stehts wissen mit wem sie gerade Informationen austauschen. Der ACME Server fungiert dann auch gleichzeitig als Anfragemöglichkeit für jeden Gesprächspartners welcher das Zertifikat des jeweils anderen Gesprächspartners prüfen möchte. Weiterführend wäre es auch Sinvoll diese Projekt statt in Pebble in letsencrypts, oder einen anderen, vollwertigen ACME Server 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. Abgerufen 1. November 2017 von https://github.com/tompollard/phd_thesis_markdown/tree/v1.0

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

[16] 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

[17] 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

[18] 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

[19] 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

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