% Abschlussarbeit

% Autor_in

% Prüfer_in

% Betreuer_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

Betreuer:
Dr. , Firma GmbH

01.03.2018

Erklärung

Richard Reik, geb. 27.08.1998 (IF8, SS 2021)

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

Zusammenfassung

(Worum geht es?) Sicherheit ist ein Thema das zunehmend an Wichtigkeit gewinnt, vor allem im Internet wird kommunikation zunehnemend verschlüsselt. Nachrichten die heutzutage unverschlüsselt verschickt werden sind nichtmehr die Regel sondern stellen eine Ausnahme dar. Daher ist es wichtig zu verstehen wie Sicherheit gewährleistet werden kann, auch wenn Geräte sich nichtmehr in eigener Hand befinden.

Zielstellung dieser Arbeit ist verwalten von Zertifikaten, welche benutzt werden können um jene Sicherheit zu gewährleisten. Dabei soll besonderenDie sichere Erstellung, Aktualisierung und Speicherung von Zertifikaten. Problemstellung: Forschungsfrage:

(Wie bin ich vorgegangen?) Methoden:

(Was sind meine wichtigsten Ergebnisse?) Ergebnisse/Fazit:

(Was bedeuten meine Ergebnisse?) Diskussionsgrundlage/Empfehlung:

Das ACME Protokol wird verwendet um es Servern zu ermöglichen

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

JOSE:

JWS:

JWT:

JWK:

Replay-Nonce:

API	**A**pplication **P**rogramming **I**nterface
JSON	**J**ava**S**cript **O**bject **N**otation

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 zentraler 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 u.a. private Daten abfragen 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 das 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. Wäre 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 deren aktualisierung. Es muss ein Gleichgewicht zwischen Funktionalität, Sicherheit sowie Verwaltbarkeit geben. Das fängt vor dem ausstellen des Zertifikates an und hört erst auf, wenn das Zertifikat sicher verwahrt wurde.

1.3 Verwandte Arbeiten

Diese Arbeit basiert auf dem ACME Protokoll, welches im RFC8555 definiert wurde, sowie einem Projekt von Google welches sich "go-attestation" nennt. In dieser Arbeit wird stark auf die Erweiterung des ACME Protokolls eingegangen, zwei Projekte die das auch getan haben, sind die Erweiterung des Verfahrens um IP Addresse, sowie das Verfahren zur Verwendung von TLS. Ich weiß nicht was ich hier sonst noch schreiben soll ...

1.4 Übersicht über die Bachelorarbeit

Diese Bachelorarbeit ist in 4 Teile eingeteilt. Ein Grundlagenkapitel in dem die wichtigsten Begriffe die in dieser Arbeit vorkommen erklärt werden, sowie eine Einführung in das ACME Protokoll. Im Kapitel 3 wird die Theoretische Umsetzung einer neuen Challenge für das ACME Protokoll besprochen. Hier wird auf die Verwendung verschiedener Schutzmaßnahmen von Gerät und Nutzer eingegangen. Im Anschließenden 4ten 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 könnte 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

2.1 TPM

Das TPM ist ein Modul mit dem Funktionen der Sicherheit auf einer physischen Ebene umgesetzt werden sollen. Auch wenn dieses Modul erstmals 2009 von der International Organization for Standardization (ISO) und International Electrotechnical Commission (IEC) [1] beschrieben wurde, existierten bereits 2011 300 Millionen dieser Chips [2]. 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[3]. Die Idee des TPM Moduls ist dabei die Limitationen und Probleme die sicherheits Software 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[4]. 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[2]. Zum anderen eine kryptographische Einheit welche Schlüsselpaare erstellen kann [5]. Dabei wird unter anderem sichergestellt, dass die kryptographischen Verfahren auf Hadware ausgeführt werden, die dafür gemacht sind 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 gesi-

cherten 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. Dabei handelt es sich um drei Teile: einen Privaten Key im gesicherten Speicher auf den nicht zugegriffen werden kann, einen public Key auf den zugegriffen werden kann, sowie ein Zertifikat welches vom Hersteller signiert wurde[6]. An diesem Zertifikat in Verbindung mit dem entsprechenden private Key kann der TPM Chip eindeutig identifiziert werden.

2.2 ACME

ACME steht für *A*utomatic *C*ertificate *M*anagement *E*nviroment, also eine Automatische Zertifikats Verwaltungs Umgebung. Diese besteht aus zwei Teilen: erstens einer CA welche Zertifikate erstellt und verwaltet 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 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 zu nutze gemacht werden. Auch hier soll mit einem automatisierten Verfahren erst der Client geprüft und anschließend ein Zertifikat ausgestellt werden. Der ACME Server arbeitet dabei mit einer Datenbank in der jeder Client der ein Zertifikat anfragen möchte erstmal einen Account erstellen muss. Für diesen Account 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 HTTPS kann der Client dabei frei wählen. Die folgenden Erklärungen beziehen sich auf ACME wie es im RFC-8555 definiert ist [**rfc-8555?**].

2.2.1 Hintergrund

Ein ACME Server der nicht gleichzeitig als Webserver dient muss laut RFC-8555 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 Account 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[7]. 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 erhalten des Zertifikates dargestellt werden. Der grundsätzliche Ablauf des ACME Protokolls ändert sich abhängig davon ob der Client bereits über ein Account auf dem Server verfügt. Hier wird jedoch davon ausgegangen, dass 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 jedem Response des Servers mitgeschickt, damit der Client nicht wieder eine neue Anfragen 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.

```
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"
```

Abbildung 2.1: RFC-8555 Get Nonce beispiel

2.2.2.2 Account erstellen

Jede Order wird fest an einen Account gebunden und so muss zu Beginn ein neuer Account erstellt werden. Dazu sendet der Client in seiner JWS Payload Mailadressen, die mit diesem Account 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. Da noch keine KID vorhanden ist wird hier im Header an dessen Stelle der JWK mit dem entsprechendem Öffentlichen Schlüssel der zum Signieren verwendet wurde an den Server

gesendet. Der Server antwortet in der Payload mit dem Status des Accounts, sowie mit der Account URL, welche im folgenden Ablauf als KID fungiert.

```
POST /acme/new-account HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
  "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"
}
```

```
HTTP/1.1 201 Created
Content-Type: application/json
Replay-Nonce: D8s4D2mLs8Vn-goWuPQeKA
Link: <https://example.com/acme/directory>;rel="index"
Location: https://example.com/acme/acct/evOfKhNU60wg

{
  "status": "valid",

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

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

Abbildung 2.2: RFC8555 New Account Server Response

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. In diesem Array wird nicht nur definiert mit welchem Verfahren (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 in im type des identifier den Wert "dns" mit. Zusätzlich kann der Client bestimmen für welchen Zeitraum das Zertifikat gültig sein soll durch das übersenden von "notBefore" und "notAfter" Werten, 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.

```
POST /acme/new-order HTTP/1.1
Host: example.com
Content-Type: application/jose+json

{
  "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"
}
```

2.2.2.4 Challenge aktivieren

Für den Client gibt es mehrere Arten auf die der Server validieren kann, dass er tatsächlich ist wer er vorgibt zu sein. Im RFC-8555 Dokument werden dabei zwei näher erläutert, die auch hier ausführlicher besprochen werden sollen. Die HTTP, sowie die DNS Challenge. Weiterführend gibt es Erweiterungen für das Dokument wie eine Challenge die die IP Adresse prüft [8] oder die Domain über TLS prüfen [9]. Um zu verstehen wie ACME funktioniert, reicht es jedoch sich auf die DNS sowie die HTTP Challenge zu beschränken.

Der Client sendet nun einen POST-as-GET Request an den Server, an die URL dessen challenge er als erstes bearbeiten möchte. Der Server antwortet nun mit Informationen zu dieser Challenge wie dem Status, wann sie abläuft, dem 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 Account Key einen Authorisierungsschlüssel (Authorization Key) erstellen. Dieser Schlüssel wird anschließend mit dem SHA-256 Verfahren verschlüsselt (hashed). Dieser Wert wird nun base64 encoded und in einem TXT Dokument gespeichert. Dieses Dokument wird dabei unter der im Identifier Value angegebenen Domain unter dem Prefix "_acme-challenge" abgespeichert. Für "www.example.org" wird die Datei also unter "_acme-challenge.www.example.org" abgespeichert [**rfc-8555?**]. Der Client sendet nun einen POST-as-GET Request zum Server um diesen zu informieren, das dieser die Datei 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, frägt von der Domain das TXT Resource Record oder die HTTP Ressource an und überprüft dass der erhaltene Wert mit dem eigenen Werte übereinstimmt. Ist das gelungen, gilt die Challenge als bestanden.

2.2.2.6 Zertifikat Management

Ist der Status für diese Order als Valide gesetzt, also wurden alle Challenges erfüllt, so kann der Client sein Zertifikat anfragen. Dazu sendet er dem Server eine 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, 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 Certifiat im Body der Antwort.

2.2.2.7 Weitere mögliche Schritte

Damit ist die Kommunikation abgeschlossen. Der Client kann nun über den erstellten Account die Zertifikate aktualisieren lassen, ohne die Challenges noch einmal durchlaufen zu müssen. Der Client kann den Server bitten den Account zu löschen oder ein Zertifikat zu widerrufen. Auch ein sogennter Key Change ist möglich, wenn der Öffentliche und Private Schlüssel, welche für den Account und damit auch für die Kommunikation mit JWS verwendet wurden, geändert werden sollen. Dabei muss der Cleint den neuen Schlüssel in einer Nachricht verpacken, welche von dem alten Schlüssel signiert wurde.

3 | Theoretische Umsetzung

3.1 Aufbau der neuen ACME Challenge

Wie im Grundlagen Kapitel 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 gelten, wie bei jeder anderen ACME Kommunikation. 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, in jeder Kommunikation des Servers mit dem Client die über anfragen eines Directorys hinausgehen sollen sie mitgeschickt werden. Alleine durch diese Maßnahme können gefahrenquellen wie Replay-Angriffen entgegengewirkt werden. Der Im Grundlagenkapitel beschriebene Aufbau in dem erst ein Account erstellt wird, 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 Website oder einen DNS prüfen.

3.1.1 Vorbereitung

Gegensätzlich zu den beiden 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 Public Key des *E*ndorsement *K*ey 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 Account erstellen und verwalten

Das im Grundlagenkapitel beschrieben Anfragen des Directorys, sowie die Verwendung von Replay-Noncen oder das erstellen des Accounts haben sich weder Clientseitig 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 Accounts durch den TPM Chip. Jeder Account 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 auch bei der 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 *A*ttestation *K*ey wie er hier verwendet wird, ist ein Container der neben einem öffentlichen Schlüssel auch Metadaten, wie die TPM Version neben

einigen anderen, besitzt. Der AK dient nicht nur als normaler Schlüssel sondern vielmehr als eine Verpackung für Informationen über den TPM Chip, Informationen die zum erstellen des 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 Geheimniss, dass durch die Verwendung von EK und AK erstellt wird zu verifizieren. Das dabei verwendete Verfahren basiert auf einem Projekt von Google zur identifizierung von Geräten [10]. Durch dieses Verfahren wird 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 nein 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 Geheimniss 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 Geheimniss 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, da dieser über einen eingebaute kryptographische Funktion verfügt und weil der Private Key nie außerhalb des Chips existieren darf. Der Public Key wird mit anderen Daten, die in der CSR stehen sollen mit dem TPM Chip verschlüsselt. Diese Nachricht wird 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 welcher zur Erstellung des Zertifikates verwendet wurde ist dabei der gleiche, welcher mit dem EK AK Verfahren überprüft wurde. Es handelt sich um den öffentlichen sowie dem privaten Teil des AK.

3.1.5 Folge Anfragen

Der Client hat nun dem Server gegenüber bewiesen, dass dieser tatsächlich die Kontrolle über den TPM Chip, welchen 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 Zertifikat verknüpft, welches den Client bereits validiert hat. Es handelt sich dabei um die Verkettung von Zertifikaten[**chain-of-trust?**]. Durch die Verknüpfung der Anfrage auf ein neues Zertifikat mit dem alten Zertifikat bewiesen dass der Client die Kontrolle über den Chip immer noch besitzt.

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 Requests abzufangen. Es soll also eine zusätzliche Sicherheit geschaffen werden um das Client Gerät vor seinen eigenen Nutzern zu schützen. Eine dem entsprechende Maßnahme ist, dass der Private Key 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 nein 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 Datenbank sowie der Bearbeitung der Challenge erweitert. Dabei soll es einem Systemadministrator einfach gemacht werden die Liste der bekannten EK Public Keys 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 Account 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. So ist es auch

einfacher Zertifikate zu widerrufen, sollte das Gerät als gestohlen oder vermisst gemeldet werden, da nun diesem Datenbank Eintrag, mit all seinen verknüpften Informationen nicht mehr vertraut werden kann.

4 | Praktische Umsetzung

4.1 Architektur

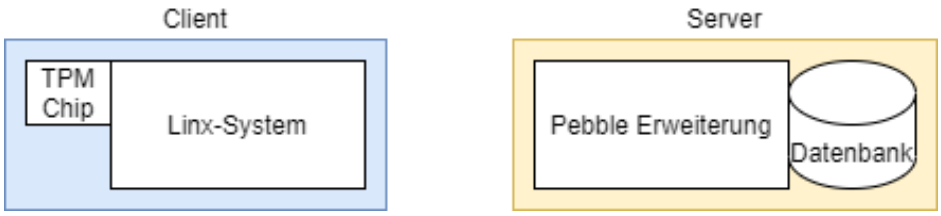

Abbildung 4.1: Vereinfachte Darstellung des Client Server Aufbaus

4.1.1 Einrichtung des ACME Client

Für die Einrichtung des ACME Client sind zwei Voraussetzungen notwendig. Einmal dass der Client auf einem Linux System läuft, welche Linux distribution dabei verwendet wird ist egal. Und andererseits die korrekte Installation des TPM Chips. Hierbei gibt es zwei Möglichkeiten wie der Chip installiert werden kann. So ist es möglich den Chip Physisch an dem jeweiligen Gerät, entsprechend seiner Anleitung 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. Hierbei muss jedoch beachtet werden dass die Sicherheit wie in den vorangegangen Kapiteln beschrieben, nur durch den physischen Chip garantiert werden kann. Da der Chip immer nur eine Kommunikation gleichzeitig erlaubt muss sichergestellt werden, dass kein anderes Programm gera-

de mit diesem Kommuniziert und eine Kommunikation so blockiert. Ein einfacher Test in Linux sieht folgendermaßen aus

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

Antwortet der TPM Chip mit "resource is busy" blockiert ein anderer Prozess. Vor allem bei Geräten auf denen bereits mit dem Chip gearbeitet wurde muss darauf geachtet werden. In der hier beschrieben Praktischen Umsetzung wurde ein Physischer TPM Chip und keine Emulation oder Simulation verwendet.

4.1.2 Aufsetzen des ACME Servers

Hierfür wurde sich bereits existierender Software bedient. Pebble ist ein ACME Server der von letsencrypt zur Verfügung gestellt wird[11]. 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. Wichtig ist hierbei nur das 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-----`
```

4.2 Implementierung

Das Projekt wurde sowohl Serverseitig wie auch Clientseitig in GO geschrieben. Als Server wurde mein eigener Rechner Verwendet, der Client lief auf einem Raspberrypi, der Code für beide wurde vorrangig auf dem Rechner geschrieben. Die IP Adressen sind beispielhafte Werte. Hierbei soll aufgezeigt werden, 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 bereitgestellt 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.

```
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
}
```

In dieser Methode wird zuallererst geprüft ob bereits einen Nonce, hier globNonce genannt, vorhanden ist. Falls nein, wird ein Request ausgesendet und der Response Header für die Replay-Nonce ausgelesen. Der erste Test ist deshalb wichtig, da im Folgenden Ablauf jede Antwort des Servers eine neue Nonce übersandt, welche den Wert von globNonce annimmt. 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 Sinnvoll das Schlüsselpaar vom TPM Chip generieren zu lassen. Beispielcode aus der Methode für den Account erstellungs 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)
}
```

Nur dieser Request wird der Öffentlichen Schlüssel verschickt. Im späteren Verlauf, sobald der Account 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 einen neuen Account zu erstellen, sondern kann auch den entsprechenden Öffentlichen Schlüssel diesem Account 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. Dazu stellt der Client eine Verbindung mit dem Chip her und liest so den Public Key des EK aus und lässt sich einen AK erstellen. Für beide Funktionalitäten kann auf das Projekt "go-attestation" [10] 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.


```
Body of newOrder Request:  {
   "status": "pending",
   "expires": "2021-08-24T13:32:22Z",
   "identifiers": [
      {
         "type": "ek",
         "value": "{\"KeyEncoding\":2,\"TPMVersion\":2,\"Public\":\"AAEACwAFAHIAAAAQABQACwgAAAAAAAEAvBUcYjceMTSIwdO4x0GL
62FPsPENJybFJ55F/Wrstgav059B2T93b6GcmXO07tr3rqYXVwY9OmnQL1PjUdna6fArLmFonuAj9V/a7b4nUz3KpLNNNOJ36oeHucrGrctkm2qhXFtbEoOo
590KYu5lvlrPUtuXVR9hcdEBd+es8Efv7Wb80XpKC2YdaKINlOZhKJdogoHGwzogBe5ygTY+FNpwuuingkD5a0B7+6oi8Ro/nCB8if/BKXomK5xz3FBqH4LS
9ga/fUxxc95WrfkiqqSnDlN7Lj0uYqNJZUTfARarim8rP4PuUn2BlAh0YOV9sQxgfXST+i/Z2GjOYAoPWw==\",\"CreateData\":\"AAAAAAAg47DEQpj8
HBSa+/TImW+5JCeuQeRkm5NMpJWZG3hSuFUBAAsAIgAL3WVUcd/LP+TLSwYd0gKdfJywZMe/2004SrJIgY6AzSkAIgALO57Ujqu4kZDgwkmA/JYEh+sFOZH+
nGjduGCxmUQl7WgAAA==\",\"CreateAttestation\":\"/lRDR4AaACIAC+rPuaHPMl06vqLxFpUf+WOP3Rdj/E6v80sO3yITkFbXAAAAAAAAJaAPbNK6j
e+xdVUsAZb+xvHWaTkhACIAC0QtRIFnBSXbpYhl/KUaHlS9My74orGz40wQT/uti/ypACBD93t2ieV9GYkZgflA23yMXbrqEVTFs4hJ5ktJ4PF/EA==\",\"
CreateSignature\":\"ABQACwEAiI/Yw9npYeJFUyakNBiImBBLHSnW0NfTrAXDymp03Ke53IyWgF4tTnVlB+oqqGwlm9GcqNwVgy2/rE/n3AS2q5zaqHte
9TsRMs37mOzKd7G2Hl7o/dY0zjbBTuTKVQFM9n7LmRs0FDGcziP29mBDh+owUJl3qhzupkrK4K+QGG8NVTqCRzlXOnptULDiic5Q3wi0TOgd6+LYleOOH5YX
ljKJO2RM/5epBqznU/aHNz+p7v4Yts5O0JsFU95i4NZnY+l+EuipOIbdwn5Z72pApObma4LXuYVPJRdBDN3vkMdkC50S0SiOTQCgtvpXgXBre9MaOJgnhPAI
gYg8LCGTuw==\",\"Name\":\"\",\"KeyBlob\":\"ACBc5tGx6TWFEeDaIqFnzWbtTyraUYlrug6wjpSMTVc4ugAQq5D4qu9F5+YKCTyHNne84HXRiVPUi
5Jlh31W3RllsS05HNefQcSBoEljhDnD0BF8gfgSj+ZCOjKDPyLnv0P9elgMsTaDtNO7r7c8nYWOcJaXQ8hTzN2jKWSlE/sUb7N1T53FwlfGoeP4kCbeYk+N2
lidukSpEn89lju8/TgzLK7SI3Zyt9ZbG5Yh2VRASqHG0ETcISbUKSRZzA==\"}-----BEGIN RSA PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAA
OCAQ8AMIIBCgKCAQEAh2oOFWso2nWgrA/6SIcJ\nxznL4ZHwlrVnphcqYVChhzC8tXdZ6eZmPWbIP4xgKtZsYSAkPbolLf3dPFlA+G5W\nxuXpE5QRnlbIo3
Rx0CxLwduy/z7Eak8HNI32eblU2jPYqCMCeLRStNjNnqZEoji4\n//cqsslBlpXWJCH8VckfpSiXBvA+0Jyk5ceY83VCVYoKBwLVhRnTEFI2TeWUOFDn\nl3
6c85//Yd+Mohx9aoTyYTiC84ePO/sJoNdKaFl8JjgsqxYPFxcCguzeCacvA/Jr\nPs853EG0Sl52FuBj2lCeB8QJUrNpabT/kFM9kBW6HQvWEgASvO0FTJ42
lCx80Ecv\nmQIDAQAB\n-----END RSA PUBLIC KEY-----\n"
      }
   ],
   "finalize": "https://192.168.1.8:14000/finalize-order/dn3VRQ7wgqoe7Gu6xnAZQCdRRAbeyKWqoqnloxYSuWs",
   "notBefore": "2021-08-01T00:04:00+04:00",
   "notAfter": "2021-08-08T00:04:00+04:00",
   "authorizations": [
      "https://192.168.1.8:14000/authZ/nQxyaQV942uQoTlAFeMFRaFpGQUZ7MiZB_HmK-xAJK4"
   ]
}
```


Abbildung 4.2: getOrder Body

So sieht der Request des Get Order requests aus, welcher an den Server gesendet wird. Um mit dem neuen Identifier etwas anfangen zu können muss der Server um den Identifier, sowie die neue "ek-01" Challenge erweitert werden:

```
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"
)
```

```
const (
    StatusPending     = "pending"
    StatusInvalid     = "invalid"
    StatusValid       = "valid"
    StatusExpired     = "expired"
    StatusProcessing  = "processing"
    StatusReady       = "ready"
    StatusDeactivated = "deactivated"

    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"
)
```

Abbildung 4.3: Erweiterung der Challenge datenbank des Servers

Nun kann bereits die erste Prüfung stattfinden. Ist der EK Wert dem Server nicht bekannt, stimmt also nicht mit dem Hart Codierten 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 einen korrekten öffentlichen Schlüssel des EK Werts zu Verfügen.

4.2.2.2 Challenge aktivieren

Der Server erkennt, dass es sich um eine "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 Geheimniss, folgend 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 Secret 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)
}
```

das "b" vor public, createData, CreateAttestation und CreateSignature welche aus AK extrahiert wurden gibt dabei an, dass es sich jeweils um byte Werte handelt. Das so erstellt Secret wird nun dem Client auf seinen Post-as-Get Request zum erhalten der Challenge zur verfügung gestellt. 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 lässt, muss hier der Client selber das entschlüsselte Geheimniss 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 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 Key den Chip nicht verlassen darf, ist es notwendig, das der CSR über den TPM Chip generiert wird. Informationen, wie der Name oder die Mail adresse sind natürlich selbst ausfüllbar. Ein Request wird nun an den Server gesendet mitsamt der CSR, 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, dass der Client jetzt noch zu erledigen hat, ist es 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 habe ich auf meinem Rechner den ACME Server und auf einem Raspberrypi mit TPM Chip den Client laufen lassen. Ziel des Ablaufes war es sicherzustellen, dass die in 4. beschrieben Client und Serverseitigen Erweiterung wie geplant funktionieren. Darunter gehören Funktionalitäten, wie ein Account anlegen und eine Certifikat einzuholen. Genauso wie die neue Challenge, die neue Art CSR zu erstellen, sowie der richtige Umgang mit dem TPM Chip und Zertifikaten

```
pi@raspberrypi:~/ACMEclient $ sudo ./comp
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

HTTP result status:  200 OK
authChallengeAnswer: Challenge answer was send!

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
```

ACME Ablauf auf Pi

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.

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 Account Key, in einen Authorisierungsschlüssel um. Bei der DNS Challenge wird zusätzlich noch mit dem SHA-256 Verfahren der Hashwert gebildet. Anschließend werden die entsprechenden Werte base64 codiert und als HTTP Resource 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 erfolgreich 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 Geheimniss, welches der Client anfragen und lösen muss. Ist das geschafft übersendet der Client das gelöste Geheimniss 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, Kontrolle über etwas beweisen. In jedem muss beweisen 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 System Integrität zumindest Teilweise zu überprüft, was im 5.5 besprochen werden sollen, gibt es keinerlei Möglichkeiten für den Server sicher zu stellen,

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. Wenn beispielsweise mehrere Webserver verwendet werden, muss sichergestellt werden das auf allen der Authorisierungsschlüssel existiert [12]. Ein weiteren Unterschied stellt die Art der Schlüssel generierung 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 Angriffs Vektoren

In diesem Kapitel sollen allgemein mögliche, sowie EK-Challenge spezifische Angriffsvektoren besprochen werden, welche die neue Challenge mit all ihrer Infrastruktur neu dazu gekommen sind oder aus dem generellen Aufbau des ACME Protokolls entstehen. Wenn mögliche Gegenmaßnahmen bekannt sind werden sie jeweils dazu geschrieben.

Wie im letzten Kapitel bereits besprochen Prüft der ACME Server nie die Integrität des entsprechenden Systems. Schafft es ein Angreifer sich die Kontrolle über den TPM Chip zu erlangen, kann er sich über ACME Zertifikate beschaffen. Zwar kann sich der Angreifer eventuell auf den Chip zugreifen, kritische Informationen wie Private Schlüssel aus dem Chip zu extrahieren ist jedoch nicht möglich.

Ein Serverseitiger Angriff kann ein *D*enial *o*f *S*ervice kurz DoS Angriff sein. Hierbei wird der Server lahmgelegt durch eine übermäßige Anzahl an Anfragen. So können zwar keine Informationen extrahiert werden, das ausstellen von Zertifikaten aber auch die Verifikation bereits vorhandener Zertifikate kann dabei jedoch lahmgelegt werden. Da der ACME Server nicht nur als Website sondern auch als CA fungiert, kann je nach Architektur des Servers, durch einen solchen Angriff großer Schaden angerichtet werden. Wenn ein neuer Kommunikationspartner auftritt kann jedes Gerät zur überprüfung dieses Zertifikates 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 [13] [14] [15].

Es gibt noch einige andere Angriffsmöglichkeiten, die nur kurz angesprochen aber nicht länger behandelt werden sollen, da sie unwahrscheinlich 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. Ohne Chip kein privaten Schlüssel, ohne privaten Schlüssel keine verschlüsselte Kommunikation. In diesem Fall muss mindestens der Chip sowie der entsprechende Eintrag in der Server Datenbank 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 Ideen besprochen werden die optional sind, oder thematisch nicht gepasst haben. Dabei soll unterschieden werden zwischen Client

seitigen 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 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 Hash verfahren 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 [16]. 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 neuen Account anlegt, kann sein Account gebundener 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 revoked werden, so kann der Server durch die Verknüpfung zwischen den Account Daten und dem EK nicht nur genau sagen welcher Client Account zu dem verlorenen Chip gehört und Auskunft darüber geben was dieser in der letzten Zeit für Anfragen gestellt hat durch die logs. Es ist ihm auch möglich sofort alle zugehörigen Zertifikate zu revoken, da diese durch die Verknüpfung mit der EK alle eindeutig sind.

Wie bereits besprochen wurde für die Umsetzung keine Richtige Datenbank verwendet. Eine Möglichkeit den Server Sinnvoll zu erweitern wäre das hinzufügen dieser mit entsprechender Infrastruktur, sowie dem bereitstellen einer API welche

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[17] laufen gelassen werden. Abgesehen davon dass der Container alle vom Server benötigten Ressourcen zur Verfügung stellt, kann so ein Server status 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 Zertifikate auch Certifikat Chaining genannt[**chain-of-trust?**]. Vereinfacht 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.

Draft als Grundlage: Ist das Draft ein fester bestandteil des ACME Protokolls kann darauf aufgebaut werden. Ein paar der möglichen Einsatzgebiete 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.

Literatur

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

[2] 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

[3] 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

[4] 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

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

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

[7] 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

[8] 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

[9] 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

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

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

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

[13] 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

[14] 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

[15] 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

[16] 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

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