

% Abschlussarbeit

% Autor\_in

% Prüfer\_in

% Betreuer\_in

---

Fakultät für  
Informatik und  
Mathematik



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

Richard Reik

Bachelorarbeit Informatik

Prüfer:

Prof. Dr. , 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 zunehmend verschlüsselt. Nachrichten die heutzutage unverschlüsselt verschickt werden sind nicht mehr die Regel sondern stellen eine Ausnahme dar. Daher ist es wichtig zu verstehen wie Sicherheit gewährleistet werden kann, auch wenn Geräte sich nicht mehr in eigener Hand befinden.

Zielstellung dieser Arbeit ist Verwalten von Zertifikaten, welche benutzt werden können um jene Sicherheit zu gewährleisten. Dabei soll besonders die 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 Protokoll wird verwendet um es Servern zu ermöglichen

Tabelle 1: Informationsverlust in %, realer Datensatz

k	l	t	Informationsverlust
2	-	-	-64,288 %
3	-	-	-69,002 %
4	-	-	-69,008 %
4	3	-	-69,233 %
4	5	-	-69,652 %
4	-	0,7	-70,164 %
4	-	0,1	-70,409 %
4	-	0,01	-82,463 %

# Inhaltsverzeichnis

<b>Zusammenfassung</b>	<b>i</b>
<b>Abbildungsverzeichnis</b>	<b>iii</b>
<b>Tabellenverzeichnis</b>	<b>iv</b>
<b>Abkürzungsverzeichnis</b>	<b>v</b>
<b>1 Einführung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Problemstellung . . . . .	1
1.3 Verwandte Arbeiten . . . . .	2
1.4 Übersicht über die Bachelorarbeit . . . . .	2
<b>2 Grundlagen</b>	<b>3</b>

## INHALTSVERZEICHNIS

---

2.1	TPM . . . . .	3
2.2	ACME . . . . .	4
2.2.1	Hintergrund . . . . .	5
2.2.2	Ablauf . . . . .	5
2.2.2.1	Die erste Nonce . . . . .	8
2.2.2.2	Account erstellen . . . . .	8
2.2.2.3	Order platzieren . . . . .	10
2.2.2.4	Challenge aktivieren . . . . .	10
2.2.2.5	Challenge erfüllen . . . . .	11
2.2.2.6	CSR wird an den Server geschickt . . . . .	11
2.2.2.7	Zertifikat wird erhalten . . . . .	12
2.2.2.8	Weitere mögliche Schritte . . . . .	12
<b>3</b>	<b>Theoretische Umsetzung</b>	<b>13</b>
3.1	Aufbau der neuen ACME Challenge . . . . .	13
3.1.1	Vorbereitung . . . . .	13
3.1.2	Account erstellen und verwalten . . . . .	14
3.1.3	EK Challenge . . . . .	14
3.1.4	CSR . . . . .	15
3.1.5	Folge Anfragen . . . . .	16
3.2	Erweiterungen . . . . .	16
3.2.1	Clientseitig . . . . .	16
3.2.2	Serverseitig . . . . .	17
<b>4</b>	<b>Praktische Umsetzung</b>	<b>18</b>
4.1	Architektur . . . . .	18
4.1.1	Einrichtung des ACME Clients . . . . .	18
4.1.2	Aufsetzen des ACME Servers . . . . .	19
4.2	Implementierung . . . . .	19
4.2.1	Implementierung des regulären ACME Ablaufs . . . . .	20
4.2.2	Erweiterung des ACME Protokolls um die neue Challenge	21
4.2.2.1	Order platzieren . . . . .	21
4.2.2.2	Challenge aktivieren . . . . .	22

---

## INHALTSVERZEICHNIS

---

4.2.2.3	Challenge erfüllen . . . . .	23
4.2.2.4	CSR wird an den Server geschickt . . . . .	24
4.2.2.5	Certifikat wird erhalten . . . . .	24
<b>5</b>	<b>Evaluation</b>	<b>25</b>
5.1	Testlauf der ACME Erweiterung . . . . .	25
5.2	Ergebnisse . . . . .	26
5.3	Vergleich der EK mit der DNS und HTTP Challenge . . . . .	27
5.4	Angriffsvektoren . . . . .	28
5.5	Mögliche Erweiterungen . . . . .	29
<b>6</b>	<b>Fazit</b>	<b>31</b>
6.1	Zusammenfassung . . . . .	31
6.2	Future Work . . . . .	32
	<b>Literatur</b>	<b>33</b>

---

# Abbildungsverzeichnis

2.1	Bildunterschrift . . . . .	7
-----	----------------------------	---



# Tabellenverzeichnis

1	Informationsverlust in %, realer Datensatz . . . . .
---	--

# Abkürzungsverzeichnis

API: **A**pplication **P**rogramming **I**nterface

JSON: **J**ava**S**cript **O**bject **N**otation

JOSE:

JWS:

JWT:

ACME:

PI:

JWK:

TPM:

Nonce:

SHA-256:

base64: (abkürzung?)

TXT:

**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 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 u.a. 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 sicher gestellt 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 sicher gestellt 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, so wie dessen aktualisierung. Es muss also ein Gleichgewicht zwischen Funktionalität, Sicherheit sowie Verwaltbarkeit geben.

## **1.3 Verwandte Arbeiten**

(TODO: microsoft projekt angeben)

## **1.4 Übersicht über die Bachelorarbeit**

## 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 Betriebssystem 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 offiziellen 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 persistenten 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 Hardware 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 persistenten Speicher abgelegt wird und das Modul zu keiner Zeit verlässt. Diese Tatsache kann verwendet werden um das Gerät, welches den TPM Chip verwendet an diesem zu identifizieren. Jeder Chip verfügt über einen sogenannten Endorsement Key. 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 *Automatic Certificate Management Environment*, 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.



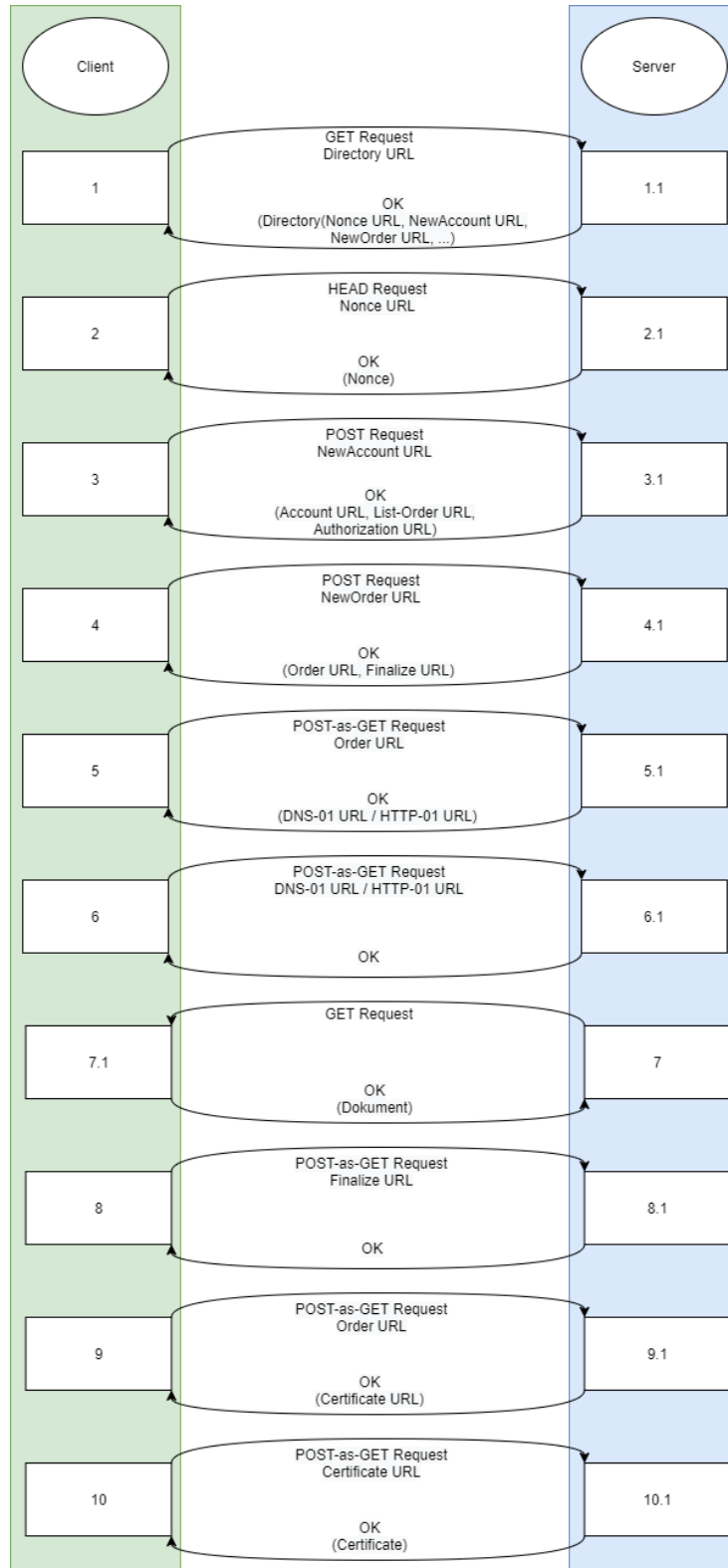


Abbildung 2.1: Bildunterschrift

### 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: oFvn1FP1wIhR1YS2jTaXbA
Cache-Control: no-store
Link: <https://example.com/acme/directory>;rel="index"
```

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": "6S8Iq0GY7eL2lsGoTZYifg",
    "url": "https://example.com/acme/new-account"
  }),
  "payload": base64url({
    "termsOfServiceAgreed": true,
    "contact": [
      "mailto:cert-admin@example.org",
      "mailto:admin@example.org"
    ]
  }),
  "signature": "RZPOnYoPs1PhjszF...-nh6X1qt0FPB519I"
}

```

RFC8555

*New Account beispiel*

```

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"
}

```

*RFC8555 New Account Server Antwort beispiel*

### 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.

### 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 [9] oder die Domain über TLS prüfen [rfc-8737?]. 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, dass 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, fragt von der Domain das TXT Dokument 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 CSR wird an den Server geschickt**

Ist der Status für diese Order als Valid 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.

#### **2.2.2.7 Zertifikat wird erhalten**

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.

#### **2.2.2.8 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.

## 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 soll die gleiche Sicherheit mit POST und POST-as-GET Requests erzeugt werden und der Ablauf soll vergleichbar sein. Auch die Replay Noncen sollen weiterhin verwendet werden. Der Klassische 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 eine Gerät Validierung und keine Domain validierung durchführen.

#### 3.1.1 Vorbereitung

Gegensätzlich zu den anderen 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 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 auslesen werden, noch angefragt werden. Diese Tatsache wird verwendet um das Gerät eindeutig zu identifizieren. Der Öffentliche Teil des EK wird Serverseitig mit einer für dieses

Gerät festgelegten Domain gespeichert. Dadurch kann der Server abgleichen ob eine Anfrage tatsächlich von einem Gerät kommt welches Serverseitig 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 nach dem Directory, 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 durch den TPM Chip. Dieses Schlüsselpaar kann für die Kommunikation mit JWS verwendet werden. 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 wäre das zwar nicht so konsequent gewährleistet, allerdings kann in der heutigen Zeit davon ausgegangen werden, dass die meisten Prozessoren in der Lage sind vernünftige Schlüsselpaare zu generieren.

### 3.1.3 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 Attestation Key wie er hier verwendet wird, ist ein Container der neben einem Öffentlichen Schlüssel auch Informationen wie die TPM Version, Create Attestation und einigen anderen besitzt. Dieser Key dient also nicht nur als normaler Schlüssel sondern vielmehr als eine Verpackung für Informationen über den TPM Chip, sowie Informationen zum erstellen eines Geheimnisses. 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. Das dabei verwendete Verfahren basiert auf einem Projekt von Google zur Identifizierung von Geräten [10].

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 Geheimniss.

Der Client kann daraufhin mit einem POST-as-GET Request dieses Geheimniss, sowie den vom Server für diesen Client zugeteilten domain Namen erfragen. Das Geheimniss kann nur mithilfe des TPM Chips entschlüsselt werden und das so gelöste Geheimniss 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" zu "processing" zu "valid". Dadurch wird sichergestellt, dass der Client die Kontrolle über den TPM Chip besitzt. Die Prüfung der Identität des Client Geräts ist also 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 eine eingebaute cryptographische 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, darunter auch dem DNS Wert den der Server in der Datenbank hinterlegt hat, 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.

---

### 3.1.5 Folge Anfragen

Der Client hat nun dem Server gegenüber bewiesen, dass dieser tatsächlich ist für wer er behauptet zu sein. Der Client kann nun, wenn eines seiner Zertifikate veraltet, einen Request zum erneuern

## 3.2 Erweiterungen

### 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 Nutzer des Client Gerätes 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 anzufordern 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, erstens 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 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 und mit DNS Namen zu versehen. 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 erstellt wurde. So ist es auch einfacher Zertifikate zu revoke, 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

#### 4.1.1 Einrichtung des ACME Clients

Für die Einrichtung des ACME Clients sind zwei Voraussetzungen notwendig. Einmal dass der Client auf einem Linux System läuft, welche Linux distribution dabei verwendet wird ist jedoch 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. Oder den Chip auf dem Client Gerät zu simulieren, welche Simulation dabei verwendet wird ist dem Architekten des Systems überlassen. Hierbei muss jedoch beachtet werden dass die Sicherheit wie in den vorangegangenen Kapiteln beschrieben nur durch den Physischen Chip garantiert werden kann. Um den Chip verwenden zu können muss sichergestellt werden das kein anderes Programm gerade mit diesem Kommuniziert und eine Kommunikation so blockiert. Ein einfacher Test in Linux ist folgender:

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

Antwortet der TPM Chip mit "ressource is busy" blockiert ein anderer Prozess. Vor allem bei Geräten auf denen bereits mit dem Chip gearbeitet wurde muss hierauf geachtet werden.

### 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. Dabei handelt es sich um eine leichte Version die für Testzwecke aber nicht auf Live Systemen verwendet werden soll. Diese Stellt alle grundlegenden Funktionen die für die Folgenden Schritte benötigt werden zur Verfügung. Einzig eine Datenbank muss hierfür erstellt werden. Welche Form diese annimmt ist jedoch wieder dem Architekten überlassen. Da es sich hier nur um ein Proof-of-Concept handelt wurden die Zertifikate, welche in der Datenbank stehen in go Programmen hartkodiert.

```
const pubPEM = `
-----BEGIN RSA PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAh2oOFWso2nWgrA/6SIcJ
xznL4ZHwlrVnphcqYVChhzC8tXdZ6eZmPWbIP4xgKtZsYSAkPbo1Lf3dPFlA+G5W
xuXpE5QRn1bIo3Rx0CxLwduy/z7Eak8HNI32eb1U2jPYqCMCeLRStNjNnqZEoji4
//cqss1B1pXWJCH8VckfpSiXBvA+0Jyk5ceY83VCVYoKBwLVhRnTEFI2TeWU0FDn
136c85//Yd+Mohx9aoTyYTiC84eP0/sJoNdKaFl8JjgsqxYPFxcCguzeCacvA/Jr
Ps853EGOS152FuBj21CeB8QJUrnPabT/kFM9kBW6HQvWEgASv00FTJ421Cx80Ecv
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.

### 4.2.1 Implementierung des regulären ACME Ablaufs

Wie bereits in den Grundlagen beschrieben bedient sich die Kommunikation zwischen dem Client und Server sogenannten Replay-Noncen sowie JWS. Um die Replay Nonce zu erhalten reicht es einen HEAD Request an die von Pebble für diesen Zweck bereit gestellt Schnittstelle zu senden.

```
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.

Zum Signieren der Nachrichten wird Clientseitig ein Schlüsselpaar generiert. 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: globPr
if err != nil {
    panic(err)
}
```

Nur dieser Request übersendet den Public key. Im späteren Verlauf, sobald der Account erstellt wurde, wird anstelle von "jwk" die "kid" die vom Server mitgeteilt wurde verwendet.

## 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 liest der Client den Public Key des EK aus dem TPM Chip und lässt sich einen AK erstellen. Für die Erstellung des AK, sowie das Auslesen des EK Wertes wird auf das Projekt "go-attestation" [10] von Google zurückgegriffen. 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. (todo: Bilder einfügen) Nun kann der Request an den Server gesendet werden. Um mit dem neuen Identifier etwas anfangen zu können muss der Server um den Identifier, sowie die neue "ek-01" Challenge erweitert werden:

---

```
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"  
)
```

*erweite-*

*ung der Pebble Identifier* Hier findet bereits die erste Prüfung statt. Ist der EK Wert dem Server nicht bekannt, steht also nicht in der Datenbank, so wird dem Client hier schon ein Fehler zurückgegeben und die Kommunikation endet hier. Die einzige Möglichkeit für den Client diesen Schritt zu meistern ist also über einen korrekten EK Wert 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 erstellte 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 der Client nun selber 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.

---

**4.2.2.4 CSR wird an den Server geschickt**

**4.2.2.5 Certifikat wird erhalten**

## 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 3. beschriebenen Client und Serverseitigen Erweiterung wie geplant funktionieren. Darunter gehören vorher bereits bekannte 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.

```
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 retrieve challenge details

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

HTTP result status: 200 OK
authChallenge: GET-as-POST request to retrieve challenge details

Value ist : "pending"
HTTP result status: 200 OK
authChallenge: GET-as-POST request to retrieve challenge 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 retrieve Certificate
```

### *ACME Ablauf auf Pi*

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 Nutzer des Client gerätes zugänglich gespeichert, sowie dessen privater Schlüssel, für den Nutzer unzugänglich gespeichert.

## 5.2 Ergebnisse

Das sind die Ergebnisse. In vitae odio at libero elementum fermentum vel iaculis enim. Nullam finibus sapien in congue condimentum. Curabitur et ligula et ipsum mollis fringilla.

## 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 jeweilige 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 DNS Eintrag zur Verfügung gestellt. Der Client sendet nun einen Request an den Server um diesen Wissen zu lassen, dass die Resource 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 diesen 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 Geheimniss korrekt gelöst gilt hier die Challenge als erfüllt.

Gemeinsamkeiten: Auffällig ist, dass alle drei Challenge Arten, Kontrolle über etwas beweisen. In jedem muss bewiesen werden, dass der Client die Kontrolle über den Wert im Identifier, egal ob EK, DNS oder Website beweisen. Dadurch dass nur die Kontrolle validiert wird, ist die Integrität des Systems irrelevant für das ACME Protokoll. Auch wenn es Möglichkeiten gibt, wie für die EK Challenge gibt, die System Integrität zumindest Teilweise überprüft werden kann, 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. Das ist eine Tatsache die im Umgang mit ACME nicht vergessen werden sollte.

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 abzufragen, 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 sicher gestellt werden das auf allen der Authorisierungsschlüssel existiert [11]. 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 welches Gerät gerade kommuniziert.

## 5.4 Angriffsvektoren

In diesem Kapitel sollen allgemein mögliche, sowie EK-Challenge spezifische mögliche Angriffsvektoren besprochen werden, welche die neue Challenge mit all ihrer Infrastruktur mit sich gebracht hat. 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 *Denial of Service* kurz DoS Angriff sein. Hierbei wird der Server lahmgelegt durch eine übermäßige Anzahl an Anfragen. So können zwar keine Informationen extrahiert werden, das austellen 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 kann großer Schaden angerichtet werden. Können Zertifikate nicht mehr verifiziert werden kann es passieren, dass Kommunikation grundsätzlich abgelehnt wird. -> Hier gibt verschiedene Möglichkeiten den Server zu schützen [12] [13] [14].

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 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 Clientseitigen Änderungen und Serverseitigen Änderungen.

Clientseitig: Wie bereits angesprochen gibt es durch den TPM Chip die Möglich-

---

keit 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 [15].

Serverseitig: Sobald ein Client einen neuen Account anlegt, kann sein Account gebundener öffentlicher Schlüssel, in der Datenbank gespeichert werden. Über sendet nun dieser Client in einem newOrder 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 revoke, da diese durch die Verknüpfung mit der EK alle eindeutig sind.



## 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 Schwerpunkt 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 vonnöten. 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 Sinnvoll 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.

Zeck der Forschungsfrage

Umsetzung

Ergebnisse

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 auszustellen 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 schwierig macht .....

## 6.2 Future Work

Auf diesem Projekt aufbauend kann nun eine

# 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](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](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>
- [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>
- [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] Challenge Typen. Abgerufen 22. August 2021 von <https://letsencrypt.org/de/docs/challenge-types/>
  - [12] 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](https://www.allianz-fuer-cybersicherheit.de/SharedDocs/Downloads/Webs/ACS/DE/BSI-CS/BSI-CS_002.pdf;jsessionid=67F80FFF524489B8981810312FE7F701.internet082?__blob=publicationFile&v=1)
  - [13] 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](https://www.allianz-fuer-cybersicherheit.de/SharedDocs/Downloads/Webs/ACS/DE/BSI-CS/BSI-CS_090.pdf;jsessionid=67F80FFF524489B8981810312FE7F701.internet082?__blob=publicationFile&v=1)
  - [14] 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](https://www.allianz-fuer-cybersicherheit.de/SharedDocs/Downloads/Webs/ACS/DE/BSI-CS/BSI-CS_025.pdf;jsessionid=67F80FFF524489B8981810312FE7F701.internet082?__blob=publicationFile&v=1)
  - [15] 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](https://trustedcomputinggroup.org/wp-content/uploads/TCG_TPM2_r1p59_Part1_Architecture_pub.pdf)
-