% Abschlussarbeit

% Autor_in

% Prüfer_in

% Betreuer_in

Fakultät für
Informatik und
Mathematik

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN
MÜNCHEN

Authentifizierung von Systemen: Beschreibung eines Verfahrens zur sicheren Generierung, Verwahrung und Aktualisierung von Zertifikaten

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 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

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

Abbildungsverzeichnis

Tabellenverzeichnis

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 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 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 das 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 vorherschen der es so schwer wie möglich macht das

Zertifkat 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 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.

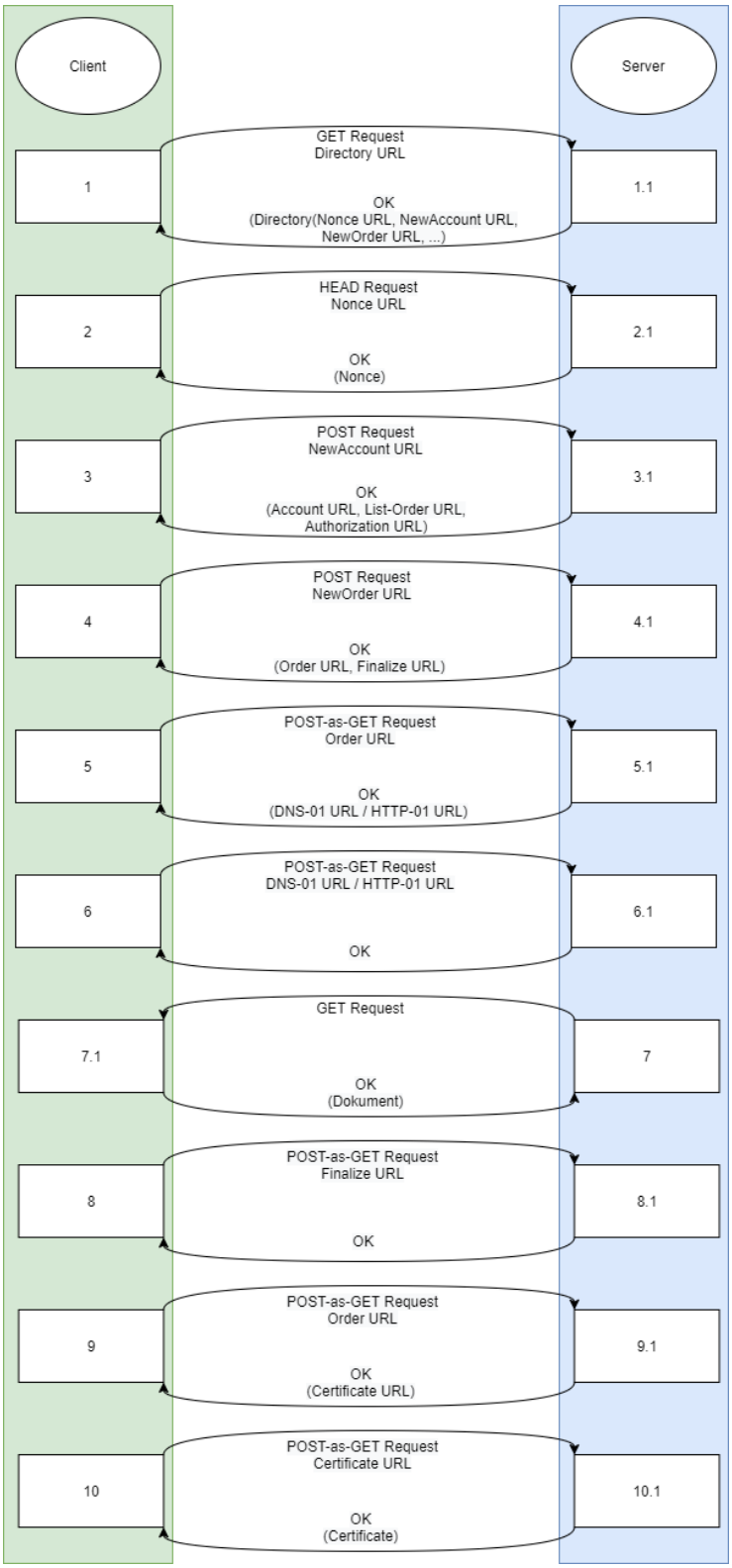

Abbildung 2.1: Bildunterschrift

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: oFvnlFP1wIhRlYS2jTaXbA
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": "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"
}
```

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, 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 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 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.

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 Certifiat 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 ausgelsen 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 registiert 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 *A*ttestation *K*ey 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 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].

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 and 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 Kontolle 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 einen eingebaute cryptographische Funktion verfügt und weil der Private Key nie auserhalb 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 gesammte 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 schüzten. 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, 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. Dadruch 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 revoken, 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 Vorraussetzungen 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 vorrangegangen 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 exisiterender 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
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 eigenener Rechner Verwendet, der Client lief auf einem Raspberrypi, der Code für beide wurde vorrangig auf dem Rechner geschrieben. Die IP Addressen 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 definerte "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 sogennte 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"
)
```

erweiterung 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 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 der Client nun 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.

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 ist es erst einen Account zu erstellen, dann eine Order mit neuem Identifier abzusenden, die neue Challenge zu verwenden und am Ende ein Zertifikat zu erhalten.

```
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

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 Vergleich der EK mit der DNS und HTTP Challenge

Vorteile und Nachteile von DNS, HTTP und EK.

HTTP und DNS Challenge: In beiden Challenges stellt der Server einen Token zur Verfügung, der die jeweilige Challenges genau definiert. Dabei wandelt der Client diesen Token, zusammen mit seinem Account Key. Bei der DNS Challenge wird zusätzlich noch mit dem SHA-256 Verfahren der Hashwert gebildet. Anschließend werden beide Werte base64 codiert 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 frägt diese Information nun ab, stimmt der Account Key mit dem vom Server generierten Wert über ein 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.

Gemeinsamkeiten: Auffällig ist, dass alle drei Challenge Arten Kontrolle über etwas beweisen. In jedem muss beweisen werden, dass der der Client die Kontrolle über den Wert im Identifier, egal ob EK, DNS oder Website nicht, besitzt.

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 Account Key abzufragen, so nimmt er in der EK Challenge eine rein passive Rolle ein. Der Server muss seinerseits keinerlei Anfragen erstellen was zu komplikationen führen könnte, wenn beispielsweise mehrere Websiten mit gleichem Namen existieren oder der DNS Anbiter keine API zur Verfügung stellt [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.

5.3 Angriffsvektoren

5.3.1 Mögliche Angriffe

5.3.2 Schutzmechanismen

5.4 Mögliche Erweiterungen

5.5 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.

5.6 Auseinandersetzung

Abbildung 2.1 zeigt wie man eine Abbildung einfügen kann. Donec ut lacinia nibh. Nam tincidunt augue et tristique cursus. Vestibulum sagittis odio nisl, a malesuada turpis blandit quis. Cras ultrices metus tempor laoreet sodales. Nam molestie ipsum ac imperdiet laoreet. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

6 | Fazit

In diesem Kapietel sollen die Möglichen Angriffsvektoren besprochen werden die bei dem DNS sowie dem IP verfahren bestanden, sowie neue die durch die neue Methode erst hinzugekommen sind. Dabei wird unterteilt in Vektoren die in allen Verfahren relevant sind, jene die speziell auf das neue Verfahren zugeschnitten sind und abschließend sollen mögiche Einfallstore besprochen werden.

6.1 Zusammenfassung

6.2 Future Work

Anhang 1: Einige Extras

Füge Anhang 1 hier hinzu. Vivamus hendrerit rhoncus interdum. Sed ullamcorper et augue at porta. Suspendisse facilisis imperdiet urna, eu pellentesque purus suscipit in. Integer dignissim mattis ex aliquam blandit. Curabitur lobortis quam varius turpis ultrices egestas.

Anhang 2: Noch mehr Extras

Füge Anhang 2 hier hinzu. Aliquam rhoncus mauris ac neque imperdiet, in mattis eros aliquam. Etiam sed massa et risus posuere rutrum vel et mauris. Integer id mauris sed arcu venenatis finibus. Etiam nec hendrerit purus, sed cursus nunc. Pellentesque ac luctus magna. Aenean non posuere enim, nec hendrerit lacus. Etiam lacinia facilisis tempor. Aenean dictum nunc id felis rhoncus aliquam.

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

[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/