FERDINAND PORSCHE
FERN FH

Ver- und Entschlüsselung am Smartphone unter Nutzung eines zentralen, sicheren Schlüsselspeichers

Masterarbeit

eingereicht von: **Mario Glaser, Bakk.**
Matrikelnummer: 00827052

im Fachhochschul-Masterstudiengang Wirtschaftsinformatik

der Ferdinand Porsche FernFH GmbH

zur Erlangung des akademischen Grades

Master of Arts in Business

Betreuung und Beurteilung: Thomas Krabina, MSc

Zweitgutachten: Dipl.-Ing. Thomas Györgyfalvay, BA MBA

Wien, Mai 2020

Ehrenwörtliche Erklärung

Ich versichere hiermit,

1. dass ich die vorliegende Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Inhalte, die direkt oder indirekt aus fremden Quellen entnommen sind, sind durch entsprechende Quellenangaben gekennzeichnet.

2. dass ich diese Masterarbeit bisher weder im Inland noch im Ausland in irgendeiner Form als Prüfungsarbeit zur Beurteilung vorgelegt oder veröffentlicht habe.

3. dass die vorliegende Fassung der Arbeit mit der eingereichten elektronischen Version in allen Teilen übereinstimmt.

Wien, 18.05.2020

Unterschrift

Kurzzusammenfassung: Ver- und Entschlüsselung am Smartphone unter Nutzung eines zentralen, sicheren Schlüsselspeichers

Die Verwendung von mobilen Endgeräten zur sicheren und vertrauenswürdigen Datenablage und Kommunikation ist nicht immer einfach. Dies macht es oft notwendig, auf die Ablage von sensiblen Daten am Smartphone zu verzichten oder das Risiko einer Kompromittierung einzugehen. Um dieses Problem zu beheben, werden in dieser Arbeit geläufige Protokolle bzw. Standards zur Authentifizierung und Autorisierung für ein Single Sign On im Einsatz auf mobilen Endgeräten untersucht. Darüber hinaus werden die Sicherheitsmerkmale von mobilen Endgeräten und zentralen Schlüsselspeichern (Hardware Security Modulen) dargestellt. Im Ergebnis ist ein Framework für die Ver- und Entschlüsselung von Daten konzipiert und experimentell für das Betriebssystem Android umgesetzt, das dem Stand der Technik bezüglich Sicherheit und Standards der Authentifizierung entspricht. Mit diesem Konzept wird gezeigt, dass die Nutzung des Smartphones keinen Verlust an Vertraulichkeit darstellt.

Schlagwörter:

OAuth 2.0, OpenID Connect, SAML, Android, Secure Element, HSM

Abstract: Usage of a central secure key storage for encryption and decryption on mobile devices

The usage of mobile devices for secure data storage or secure communication with different parties is difficult to apply. This fact makes it often necessary to avoid storing sensitive data on a mobile device or to accept the risk of compromise respectively less security. Common standards and protocols for authentication and authorization for achieving a single sign-on on mobile devices are examined in this thesis to encounter this problem. In addition, the security features of mobile devices and central key stores (hardware security modules) are outlined. As a result, a framework for the encryption and decryption of data, which corresponds to the state of the art in terms of security and authentication standards was designed and experimentally implemented for the operating system Android. This concept shows that the use of a mobile device does not represent a loss of confidentiality.

Keywords:

OAuth 2.0, OpenID Connect, SAML, Android, Secure Element, HSM

Widmung

Diese Arbeit möchte ich meiner Familie widmen, die mich auf meinem Weg durch das Studium immer wieder bestärkt hat.

Danksagungen

An dieser Stelle möchte ich mich bei allen Personen bedanken, die mich mit ihrer persönlichen und fachlichen Unterstützung bei der Erstellung dieser Arbeit unterstützt haben.

Ein Dankeschön gilt auch meinen Kommilitonen, die mich durch mein Studium begleitet haben und mir zur Seite gestanden sind.

Darüber hinaus ein herzliches Dankeschön an meine Lebensgefährtin, die mich im Alltag unterstützt hat und mir immer mit Zuversicht und Kraft zur Seite gestanden hat.

Vorwort

Die Idee für dieses Thema kam mir in meiner beruflichen Tätigkeit, in der ich stets mit der Problemstellung konfrontiert bin – wie können Daten sicher abgelegt werden und wo sind die Grenzen im Einsatz eines Hardware Security Moduls. Durch den stetig wachsenden Einsatz von mobilen Geräten und der steigenden Mobilität wollte ich dieses Thema mit dem Einsatz von Smartphones verknüpfen.

Inhaltsverzeichnis

1. Einleitung

Mobile Endgeräte sind ein allgegenwärtiger Begleiter in unserem Leben. Für eine Vielzahl an Tätigkeiten unterstützen sie uns, immer die notwendigen Informationen parat zu haben. Der Schutz unserer privaten bzw. persönlichen Daten ist ein hohes Gut, das in einer immer stärker werdenden Vernetzung eine große Herausforderung darstellt. In dieser Arbeit möchte ich einen Ansatz darstellen, wie einerseits eine komfortable Nutzung einer Anwendung mittels übergreifendem Single Sign On und einer Ver- und Entschlüsselung von sensiblen Daten aussehen kann.

In diesem Abschnitt wird die Motivation für die Bearbeitung des Themas ausgeführt und die sich daraus abgeleitete Zielsetzung bzw. Forschungsfrage beschrieben. Abschließend wird ein kurzer Überblick über den inhaltlichen Aufbau dieser Arbeit und die methodische Vorgangsweise gegeben.

1.1 Motivation

Die Verwendung von mobilen Endgeräten ist aus dem heutigen Leben nicht mehr wegzudenken. Praktisch für alle Tätigkeiten stehen uns nützliche Anwendungen zur Verfügung und unterstützen uns. Der Einsatz von Smartphones als mobile Datenspeicher oder zu Kommunikationszwecken ist, sowohl im privaten als auch geschäftlichen Bereich, allgegenwertig. Die Ablage von sensiblen Daten, wie Finanzdokumente, Verträge oder Zugangsdaten sowie Daten/Passwort-Safes – soll dabei stets unter Wahrung der Vertraulichkeit dieser Daten zur Verfügung stehen. Der Schutz der Privatsphäre endet aber nicht bei der sicheren Ablage von Daten, sondern betrifft alle Tätigkeiten, die mit dem Smartphone durchgeführt werden, bis hin zur vertraulichen Kommunikation zwischen Personen. Neben dem stets wachsenden Funktionsumfang dieser Anwendungen soll dabei niemals der persönliche Datenschutz vergessen werden.

Eine weitere Herausforderung stellt die Identifikation der Nutzer auf den mobilen Geräten dar. Damit die nützlichen Anwendungen behilflich sein können, müssen

diese die Identität der Anwender feststellen können. Dies kann über integrierte Mechanismen zur Authentifizierung passieren, bei denen wir uns über ein Identifikationsmerkmal (z.B. E-Mail-Adresse und Passwort) registrieren und zu einem späteren Zeitpunkt anmelden müssen. Immer öfter sieht man allerdings andere Anmelde-Methoden wie Login mit „Google“, „Facebook“ oder seit dem letzten Update des iOS Ökosystems – Anmeldung mit „Apple ID“. Dies bietet dem Benutzer eine komfortable Möglichkeit, sich über mehrere Anwendungen hinweg einfach anmelden zu können (Single Sign On), ohne für jedes Services ein eigenes Passwort vergeben bzw. merken zu müssen. Welche Verfahren bzw. Protokolle dafür zur Anwendung kommen und wie diese mit den Sicherheitsmechanismen der mobilen Endgeräte interagieren, bleibt dem Benutzer allerdings verborgen. Mobile Anwendungen (Apps) sind geschlossene Einheiten, die aus Sicherheitsgründen in einer geschützten Umgebung ablaufen, die Nutzer möchten aber trotzdem eine komfortable Anmeldung durchführen können.

1.2 Zielsetzung

In dieser Arbeit wird ein Framework erarbeitet, das die Ver- und Entschlüsselung von Daten am Smartphone unter der Nutzung eines zentralen und sicheren Schlüsselspeichers – Hardware Security Module (HSM) – ermöglicht.

Die Zielsetzung wird in Abbildung 1 abstrakt dargestellt. Die Authentifizierung und Autorisierung der Abläufe und Verfahren für die Nutzung soll über eine zentrale Komponente zur Identitäts-Feststellung erfolgen und dabei auf standardisierten Protokollen, die für den Einsatz im mobilen Bereich genügen, erfolgen. Nachdem sich der Benutzer authentifiziert hat, soll von einer zentralen Security-Anwendung ein Schlüssel bzw. Schlüsselpaar zur Ver- und Entschlüsselung am Smartphone bezogen und auf diesem im geschützten Bereich abgelegt werden.

Der Schutz der Schlüssel soll hierbei bei der zentralen Security-Anwendung durch den Einsatz eines Hardware Security Moduls ermöglicht werden. Am Smartphone sollen die kryptographischen Objekte durch die vorhanden Sicherheitsmechanismen

am mobilen Endgerät vor einer Kompromittierung und missbräuchlichen Verwendung geschützt werden.

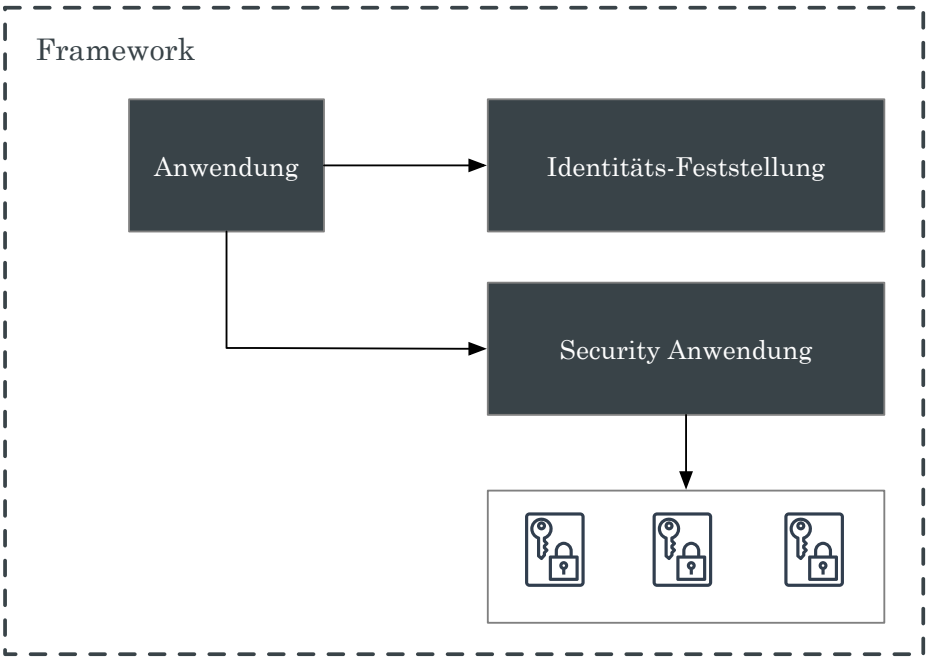

Abbildung 1: Zielsetzung

Abgeleitet aus der Zielsetzung wird in dieser Arbeit die nachfolgende Forschungsfrage beantwortet:

Wie kann ein Framework für die Nutzung eines zentralen Hardware Security Modules zur Ver- und Entschlüsselung vom Smartphone, unter Berücksichtigung technischer Sicherheitsbedenken, basierend auf Standards im Bereich der Authentifizierung und Autorisierung, aussehen?

1.3 Inhaltlicher Aufbau dieser Arbeit

Diese Arbeit ist in zwei Teile gegliedert, einen theoretischen und einen empirischen Teil, die dem methodischen Vorgehen (siehe Abschnitt 1.4) entsprechen.

Im Kapitel 2 werden zunächst die theoretischen Grundlagen für die experimentelle Umsetzung beschrieben. Dazu werden in Abschnitt 2.1 Protokolle und Verfahren die

zur Authentifizierung genutzt werden können beschrieben. Zu den Protokollen in diesem Abschnitt gehört SAML in Abschnitt 2.1.1, OpenID Connect in Abschnitt 2.1.2 bzw. FIDO in 2.1.4. Eine Besonderheit in diesem Abschnitt stellt die österreichische Lösung der Handy-Signatur bzw. die zukünftige Lösung E-ID (Elektronische Identität) dar. Diese Lösung, welche auf den Verfahren SAML (Security Markup Language) und OpenID Connect basiert, wird in Abschnitt 2.1.3 beschrieben. In Abschnitt 2.2 wird die Sicherheitsarchitektur der SOCs (System on a Chip) der mobilen Betriebssysteme und im darauffolgenden Abschnitt 2.3 das Thema Hardware Security Modul umrissen. Das Kapitel wird in Abschnitt 2.4 mit einem Fazit über die theoretischen Grundlagen abgeschlossen.

In Kapitel 3 wird auf Grundlage der Analyse im vorhergehenden Kapitel 2 eine experimentelle Umsetzung konzipiert und angewendet. Dazu wird in Abschnitt 3.1 ein Framework für die Ver- und Entschlüsselung am Smartphone entworfen bzw. die Anwendungsfälle (Abschnitt 3.1.1) und Abläufe (Abschnitt 3.1.2) dieses Frameworks definiert. Ausgehend von diesem Konzept wird in Abschnitt 3.2, anhand des konkreten Anwendungsfalls der Ver- und Entschlüsselung von Daten, eine prototypische Implementierung durchgeführt und somit die Machbarkeit überprüft.

Im letzten Kapitel 4 wird einleitend in Abschnitt 4.1 eine Zusammenfassung bzw. in Abschnitt 4.2 das Ergebnis dieser Arbeit dargestellt. Abschließend wird in Abschnitt 4.3 ein Ausblick auf zukünftige Erweiterungen sowie weitere Forschungsfelder gegeben.

Im Anhang der Arbeit werden relevante Auszüge des Quellcodes der Umsetzung (z.B. Authentifizierung in der mobilen Anwendung) aufgelistet.

1.4 Methodisches Vorgehen

Für die Arbeit wird die Vorgangsweise der konzeptionell-deduktiven Analyse mit einer prototypischen bzw. experimentellen Umsetzung gewählt. [SHB15]

Zu diesem Zweck werden vorhandene Standards, Verfahren und Gegebenheiten von mobilen Endgeräten, die aktuell in Verwendung sind, identifiziert, untersucht und qualitativ überprüft, ob diese der gegenständlichen Problemstellung genügen.

Aufbauend auf dieser theoretischen Analyse erfolgt eine Überprüfung dieser Grundlagen anhand einer konzeptionell bzw. prototypischen Umsetzung eines Frameworks zur Ver- und Entschlüsselung am Smartphone.

2. THEORETISCHE GRUNDLAGEN

In diesem Kapitel werden die theoretischen Grundlagen für eine experimentelle Umsetzung eines Frameworks zur sicheren Ablage von kryptographischen Objekten auf einem mobilen Endgerät dargestellt.

Im ersten Abschnitt 2.1 wird die theoretische Basis für die prototypische Umsetzung geschaffen. Dazu werden standardisierte Protokolle – wie OpenID Connect, SAML und andere Verfahren bzw. Standards – ermittelt und analysiert sowie für den praktischen Einsatz in der beschriebenen Problemstellung gegenübergestellt. Basierend auf diesem Ergebnis werden die Grundlagen für die Umsetzung der Authentifizierung und Autorisierung der experimentellen Umsetzung geschaffen.

Im zweiten Abschnitt 2.2 werden die technischen Sicherheitsmechanismen der aktuell verwendeten mobilen Betriebssysteme Apple iOS und Google Android analysiert, wie Daten und Authentifizierungsinformationen an mobilen Geräten vor Dritten (Personen und fremde Apps) geschützt werden können. Weiters wird – am Beispiel des Google Titan M – eruiert, welchen zusätzlichen Sicherheitsgewinn ein dedizierter kryptographischer Chip mit sich bringt.

Im dritten Abschnitt 2.3 wird ein kurzer Überblick über die wesentlichen Eigenschaften beim Einsatz eines Hardware Security Moduls gegeben.

Im vierten Abschnitt 2.4 erfolgt eine Zusammenfassung über das Fazit der theoretischen Grundlagen dieser Arbeit, insbesondere über die Protokolle SAML und OpenID, die in Abschnitt 2.1 analysiert wurden.

2.1 Analyse der Standards zur Authentifizierung und Autorisierung im mobilen Bereich

Anwendungen müssen in geeigneter Form die Benutzer authentifizieren, um diese in weiterer Folge autorisieren zu können. Oftmals werden diese Schritte direkt in der Anwendung implementiert und sind damit ein integraler Bestandteil dieser.

Stehen dem Nutzer/der Nutzerin mehrere Anwendungen zur Verfügung, muss er/sie diesen Prozess Anwendung für Anwendung durchlaufen, von der Registrierung bei der Anwendung bis zur Anmeldung selbst, womit zwei wesentliche Probleme bestehen:

Erstens müssen die „Credentials“ in jeder Anwendung sicher verwaltet bzw. gespeichert werden. Dies stellt sowohl den Benutzer/die Benutzerin vor die Herausforderung sich eine Vielzahl an Passwörter zu merken, als auch ein Unternehmen mit einer Vielzahl an bereitgestellten Anwendungen und der administrativen Prozesse eines Service Centers.

Zweitens stellt es ein sicherheitsbedenkliches Problem dar. In jeder Anwendung muss ein vergleichbares Authentifizierungsverfahren wie z.B. über Benutzername und Passwort implementiert werden. Ändert sich die Methode der Anmeldung auf eine z.B. Zwei-Faktor-Authentifizierung (2FA), muss dies in allen Anwendungen implementiert werden.

Aus diesen und weiteren Problemen heraus haben sich sowohl standardisierte als auch proprietäre Lösungen etabliert. In dieser Arbeit wird auf drei Standards eingegangen. OpenID Connect, SAML und die Handy-Signatur haben sich in Österreich als „Standards“ etabliert. Allen drei gemein ist, dass sie auf dem folgenden Schema – siehe nachfolgende Abbildung 2 - aufbauen, wobei die Abläufe, Datenformate und Bezeichnungen der Entitäten zwischen den Standards variieren.

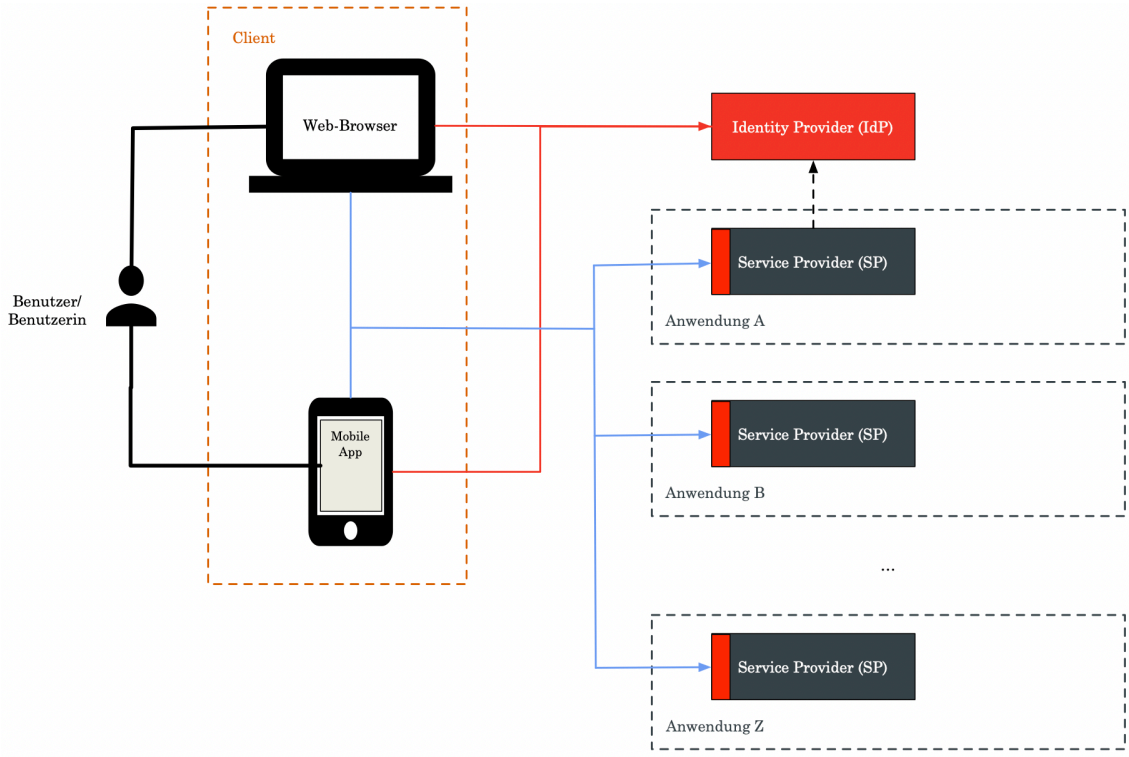

Abbildung 2: Entitäten der Authentifizierung

Der Identity Provider ist für die Verwaltung der Benutzer/Benutzerinnen (Identitäten) zuständig. Weiters gibt er auch die Methode der Anmeldung für die Anwendung vor sowie die Attribute des Benutzers.

Der Service Provider schützt eine Anwendung selbst. Er prüft dabei die Bestätigung (Assertion) der Anmeldung des Benutzers/der Benutzerin über den Identity Provider und die Zugriffsberechtigung auf die Anwendung.

Der Client führt die Anmeldeabläufe durch, dies kann ein Web-Browser, eine mobile Anwendung oder auch ein Backend-Webservice sein.

Der Benutzer/die Benutzerin möchte eine oder mehrere Anwendungen bzw. Services über die dieselben Anmeldeverfahren (Single Sign On) und „Credentials“ nutzen.

2.1.1 Security Assertion Markup Language (SAML)

Beim Thema Federation Protokolle und Identity Federation kommt man am Protokoll SAML (Security Assertion Markup Language) nicht vorbei. Dieses Protokoll war eines der ersten standardisierten Verfahren um ein Single Sign On zu ermöglichen. Es wurde von OASIS (Organization for the Advancement of Structured Information Standards) definiert und wurde in Version 2 im Jahr 2005 veröffentlicht. Im wesentlichen beruht SAML auf XML Artefakten, die zwischen Identitäten ausgetauscht werden, um Benutzer zu authentifizieren und in weiterer Folge zu autorisieren. Ein wesentliches Merkmal von SAML ist, dass sich die Begriffe die mit SAML eingeführt wurden (z.B. Identity Provider) auch bei anderen Protokollen und Konzepten wiederfinden. Die Elemente der verschiedenen Spezifikation rund um SAML sind Assertions, Protokolle und Bindings – die in ihrer Zusammensetzung anschließend Profile ergeben, wie nachfolgende Abbildung 3 veranschaulicht.

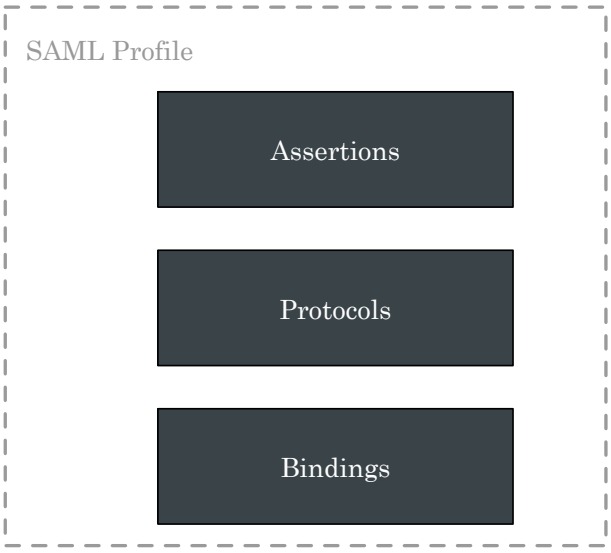

Abbildung 3: SAML Profile

Die Kombination von Assertions, Protokollen und Bindings ergeben Anwendungsfälle, in denen das Protokoll SAML eingesetzt wird.

2.1.1.1 SAML Assertions

SAML Assertions sind Aussagen, die der Identity Provider über eine Authentifizierung einer Person tätigt. Darunter fallen unter anderem: wer (Subject) hat sich, wie zu welchem Zeitpunkt angemeldet, welche weiteren Eigenschaften (Attribute) besitzt dieses Subject und für welchen Service Provider wird diese Assertion ausgestellt. Eine Assertion wird im XML Format vom Identity Provider ausgestellt und bei einem Service Provider eingelöst. In dem nachfolgendem Listing 1 ist eine SAML Assertion vom IDP (idp.example.org) für j.doe@example.org, der eine Passwort basierte Anmeldung durchgeführt hat, dargestellt [Se08]:

```
1. <saml:Assertion
2.   xmlns:saml=”urn:oasis:names:tc:SAML:2.0:assertion”
3.   Version="2.0" IssueInstant="2005-01-31T12:00:00Z">
4.   <saml:Issuer Format=urn:oasis:names:SAML:2.0:
5.     nameid-format:entity>
6.     http://idp.example.org
7.   </saml:Issuer>
8.   <saml:Subject>
9.     <saml:NameID Format="urn:oasis:names:tc:SAML:1.1:
10.      nameid-format:emailAddress">
11.      j.doe@example.com
12.    </saml:NameID>
13.  </saml:Subject>
14.  <saml:Conditions
15.   NotBefore="2005-01-31T12:00:00Z"
16.   NotOnOrAfter="2005-01-31T12:10:00Z">
17.  </saml:Conditions>
18.  <saml:AuthnStatement
19.   AuthnInstant="2005-01-31T12:00:00Z"
20.   SessionIndex="67775277772">
21.    <saml:AuthnContext>
22.      <saml:AuthnContextClassRef>
```

```
23.         urn:oasis:names:tc:SAML:2.0:ac:classes:
24.         PasswordProtectedTransport
25.       </saml:AuthnContextClassRef>
26.     </saml:AuthnContext>
27.   </saml:AuthnStatement>
28. </saml:Assertion>
```

Listing 1: Beispiel SAML Assertion [Se08]

Die Assertion beinhaltet somit folgende Elemente:

1. Welcher Identitiy Provider hat die Assertion ausgestellt *(saml:Issuer)*?
2. Wer hat sich angemeldet *(saml:Subject) bzw. w*elche Attribute (Eigenschaften) hat das Subject?
3. Ab welchem Zeitpunkt und wie lange ist die Assertion gültig *(saml:Conditions)*?
4. Wie (z.B. Passwort) wurde die Anmeldung durchgeführt *(saml:AuthnContextClassRef)*?
5. Für welchen Service Provider wurde die Assertion ausgestellt?

Um die Vertraulichkeit, Integrität und Authentizität dieser Assertions zu gewährleisten, werden diese XML Artefakte mittels Public Key Kryptographie geschützt bzw. verschlüsselt. Die dafür notwendigen Meta-Daten werden zwischen dem Identity Provider und dem Service Provider ausgetauscht.

2.1.1.2 SAML Protokolle

Nachfolgendend werden die zwei bedeutendsten Protokolle für diese Arbeit das *Authentication Request Protocol* und *Artifact Resolution Protocol* dargestellt.

Authentication Request Protocol

Über dieses Protokoll wird definiert, wie der Identity Provider und Service Provider Assertion über ein Subject austauschen. Hierbei gibt es zwei unterschiedliche Abläufe, die als (1) IDP-Initiated bzw. (2) SP-Initiated bezeichnet werden. Im

ersteren wird der Authentifizierungsablauf direkt beim Identity Provider gestartet und anschließend die Assertion beim Service Provider eingelöst. Im zweiten Ablauf wird zuerst der Service Provider kontaktiert, welcher mit einem *SAML Authn Request* antwortet. Der Client sendet diesen anschließend an den Identity Provider, welcher mit einem *Authn Response* und der Assertion antwortet.

Bei ersterem Lösungsweg (IDP-Initiated) kann die Umleitung vom Identity Provider zum Service Provider zu Problemen führen ([SM18a]). Da darüber hinaus dieser Ablauf bei OpenID (siehe Abschnitt 0) nicht definiert ist, wird in der nachfolgenden Abbildung 4 nur der zweite Lösungsweg (*SP-Initiated*) dargestellt und in weiterer Folge behandelt.

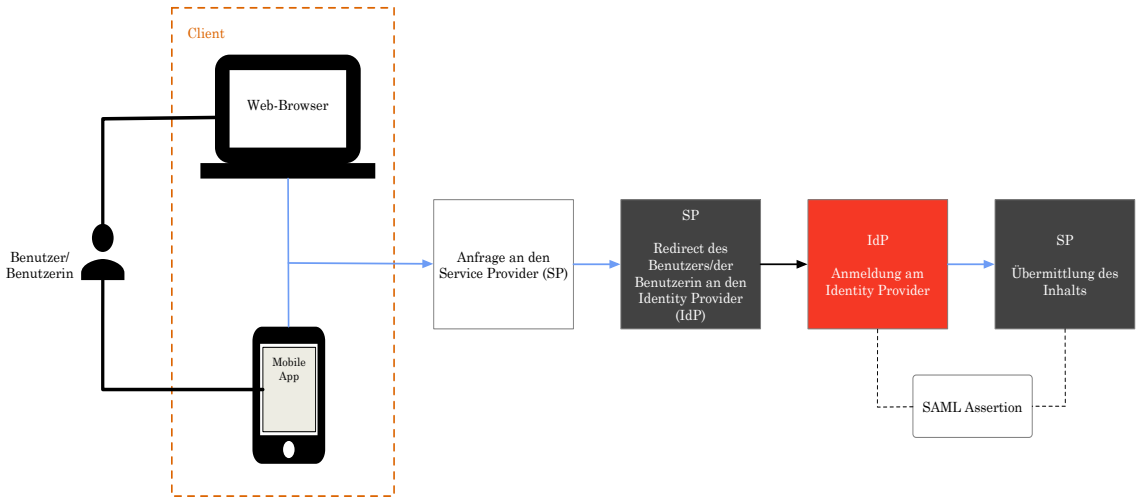

Abbildung 4: SAML SP-Initiated Authentication

Artifact Resolution Protocol

Beim Artifact Resolution Protocol werden die Assertions nicht direkt über den Client ausgetauscht, sondern der Identity Provider liefert im Zuge der Anmeldung nur einen *Identifier* an den Client. Anschließend kann der Service Provider über diese Referenz sich direkt das Artefakt vom Identity Provider holen. Diese Kommunikation zwischen Service Provider und Identity Provider wird auch als

Back-Channel Kommunikation bezeichnet und ist vergleichbar mit OpenID Connect und dem Authorization Code Flow (siehe Abschnitt 0).

Darüber hinaus gibt es noch weitere Protokolle wie z.B. das *Single Logout Protocol*, da dies aber keine Relevanz für dies Arbeit hat, werden diese nicht näher erläutert.

2.1.1.3 SAML Bindings

Um die Nachrichten zwischen zwei Beteiligten in den SAML Abläufen zu übertragen, definiert SAML sogenannte Bindings [Bi05]. Nachfolgend sind die relevanten Übertragungsabläufe für den späteren Vergleich dargestellt.

Beim *HTTP Redirect Binding* (siehe Listing 2) werden die Informationen zwischen den Akteuren über HTTP Parameter und den URL Query String (GET) über den Browser ausgetauscht.

```
1. https://idp.example.org/SAML2/SSO/Redirect?
2. SAMLRequest=request&RelayState=token
```

Listing 2: Beispiel SAML HTTP Redirect Binding [Bi05]

Dabei sendet der Service Provider im Zuge einer Anmeldung den *SAML Authn Request* (siehe nachfolgendes Listing 3) über den Client.

```
1.  <samlp:AuthnRequest
2.    xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
3.    xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
4.    ID="identifier_1"
5.    Version="2.0"
6.    IssueInstant="2004-12-05T09:21:59Z"
7.    AssertionConsumerServiceIndex="1">
8.    <saml:Issuer>
9.      https://sp.example.com/SAML2
10.    </saml:Issuer>
```

```
11.   <samlp:NameIDPolicy AllowCreate="true"
12.     Format="urn:oasis:names:tc:SAML:2.0:
13.             nameidformat:transient"/>
14. </samlp:AuthnRequest>
```

Listing 3: Beispiel SAML Authn Request [Bi05]

Eine alternative der Übertragung der Daten ist das *HTTP Post Binding* (siehe Listing 4). Bei diesem werden die Daten über den Content der HTTP Kommunikation übertragen. Die Assertion wird in den HTML Content eingebettet und über ein HTML Formular an die Gegenstelle übermittelt. In der Regel wird das Auslösen (submit) des Formulars automatisch durch ein Skript ausgelöst.

```
1. <form method="post"
2.   action="https://sp.example.com/SAML2/SSO/POST" ...>
3.   <input type="hidden" name="SAMLResponse" value="response" />
4.   <input type="hidden" name="RelayState" value="token" />
5. ...
6.   <input type="submit" value="Submit" />
7. </form>
```

Listing 4: Beispiel SAML HTTP Post Binding

Die Übertragung der Daten über HTTP Post eignet sich dann, wenn diese aufgrund der Größe nicht über die URL übertragen werden können. Dies ist beim Austausch von XML Artefakten schnell der Fall und daher kommt oftmals das HTTP Post Binding zum Einsatz.

Das *SAML Artifact Binding* definiert einen Austausch der SAML Request oder Response über eine Referenz, der Artefakt ID. Diese kann über HTTP Redirect oder HTTP Post ausgetauscht werden. Anschließend wird mittels *Artifact Resolution Protocol* und der Artefakt ID die eigentliche SAML Nachricht ausgetauscht.

Darüber hinaus existieren, wie bei den Protokollen, weitere, die nicht näher ausgeführt werden, wie z.B. der Austausch von SAML Artefakte über SOAP (Simple Object Access Protocol).

2.1.1.4 SAML Profiles

Unter *SAML Profiles* wird ein Set an Definitionen verstanden, wie SAML Assertions bzw. Nachrichten ausgetauscht und verarbeitet werden [Pr05]. Das verbreitetste Profil ist das *Web SSO* Profile, welches die Authentifizierung für HTTP basierte Clients definiert. Ein weiteres Profil zur Benutzer-Authentifizierung ist das *Enhanced Client or Proxy (ECP)* Profil.

Das *Web SSO Profile* nutzt zum Austausch der Nachrichten die im vorherigen Abschnitt dargestellten Bindings HTTP Redirect, HTTP Post bzw. HTTP Artifact für den Austausch über den Browser. Die Assertions werden dabei über den Browser zwischen den Parteien ausgetauscht. Das Profil definiert zwei wichtige Services, das *Single Sign-On Service* spezifiziert den Endpunkt des Identity Provider für den Empfang der Nachrichten des Service Providers. Das *Assertion Consumer Service* auf der Gegenseite stellt den Endpunkt des Service Providers für den Empfang der Nachrichten des Identity Providers dar.

Das *Enhanced Client or Proxy (ECP) Profile* definiert Abläufe für Szenarien, in denen der Client (Anwendung), im Gegensatz zu einem einfachen Browser, erweiterte Funktionalitäten aufweist. Im Vergleich zu den definierten Abläufen des *Web SSO Profile*, bei dem der Kommunikationsfluss über den Browser stattfindet, kann das *ECP Profile* für Anwendungen („rich clients“) eingesetzt werden. Zu diesen Anwendungen zählen z.B. mobile Endgerät nach dem technischen Stand von 2005 – aus diesem Grund sei dieses Profil an dieser Stelle nur erwähnt, da es nach den heutigen technischen Möglichkeiten auf mobilen Geräten keine Relevanz hat.

Darüber hinaus existieren noch weitere Profile, wie zum Beispiel für ein *Single Logout* in einem verteilten System mit mehreren Service Providern.

2.1.2 OAuth 2.0 und OpenID Connect

OAuth 2.0 und OpenID Connect stellen zwei Protokolle für die Autorisierung und Authentifizierung dar. Die Zielsetzung von beiden Protokollen ist die Berücksichtigung für den Einsatz im mobilen Bereich und einfach zu implementierende Abläufe zu definieren.

2.1.2.1 OAuth 2.0 Überblick

Das OAuth 2.0 Framework [Th18a] definiert ein Verfahren zur Autorisierung von HTTP Aufrufen für Dritt-Anwendungen. Die erste Version des Protokolls OAuth 1.0 [Th10] findet keine Relevanz mehr und ist nicht kompatibel mit der aktuellen Version. OAuth ist aber weniger wie ein Protokoll zu verstehen, sondern als Framework für den Einsatz bei verteilter Autorisierung und Zugriffssteuerung.

In der Version 2 von OAuth werden sogenannte Bearer Tokens verwendet, um den Zugriff auf eine geschützte Ressource abzusichern. Der Besitzer der „Ressource" gewährt Dritt-Anwendungen Zugriff ohne seine „Credentials" weiterzugeben. Dabei kommen die Rollen *Client*, *Authorization Server*, *Ressource Server* und *Ressource Owner* zum Einsatz – siehe Abbildung 5.

Der *Client* stellt hierbei die Anwendung dar, die auf die geschützte Ressource zugreifen möchte – autorisiert mit dem Bearer Token, der vom *Authorization Server* ausgestellt wurde. Die Ausstellung der Zugriffstoken erfolgt nach der Genehmigung durch den Benutzer (*Resource Owner)* für bestimmte Ressourcen (Scopes). Die geschützten Ressourcen werden vom *Resource Server* zur Verfügung gestellt, sofern ein gültiger Bearer Token durch den Client vorgewiesen wird.

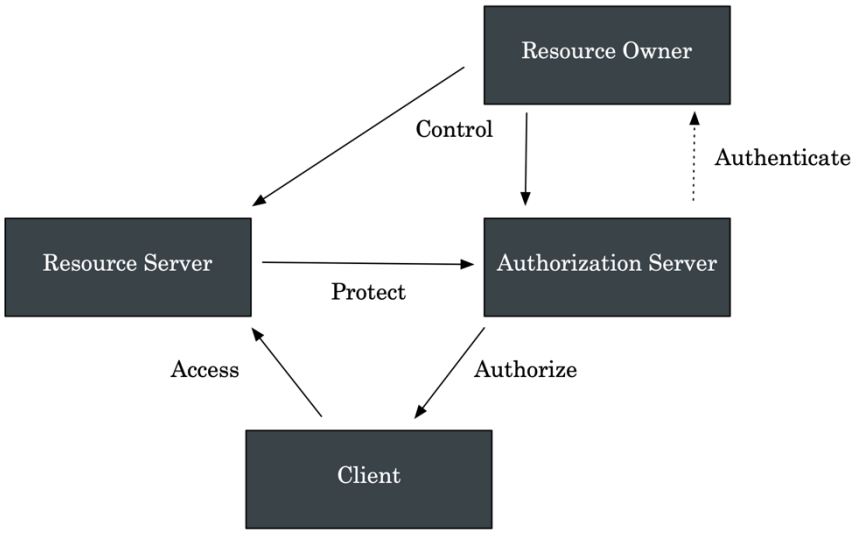

Abbildung 5: OAuth 2.0 Rollen (in Anlehnung an [SM18a])

Der *Authorization Server* erteilt die Berechtigung für den Zugriff auf eine bestimmte Ressource. Dafür werden die Clients registriert und durch eine *client_id* identifiziert. Optional, je nachdem um welchen Typ von Client es sich handelt, müssen sich diese - wie der Benutzer - beim Authorization Server authentifizieren (z.B. durch ein *secret*). Nach Authentifizierung des Clients und Bestätigung des Benutzers, die Ressourcen freizugeben, wird der Bearer Token ausgestellt und an den Client zurückgeliefert.

Der *Resource Server* ist unter OAuth vergleichbar mit dem Service Provider des SAML Protokolls (siehe Abschnitt 2.1.1). Dieser prüft, ob der vorgelegte Token noch gültig ist, für die Ressource berechtigt ist, den korrekten Scope besitzt und von einem vertrauenswürdigen Authorization Server ausgestellt wurde. Die Prüfung der Tokens kann entweder über den Authorization Server erfolgen oder direkt, um weniger Aufrufe zu anderen Stellen durchzuführen. Erfolgt die Prüfung des Status des Tokens über den Authorization Server, kann dieser nach RFC 7662 [OA15] Meta-Information über diesen bereitstellen. Das nachfolgende Beispiel (Listing 5) stellt einen Response dieser Introspektion Abfrage dar:

```
1. HTTP/1.1 200 OK
2. Content-Type: application/json
3.
4. {
5.   "active": true,
6.   "client_id": "l238j323ds-23ij4",
7.   "username": "jdoe",
8.   "scope": "read write dolphin",
9.   "sub": "Z5O3upPC88QrAjx00dis",
10.  "aud": "https://protected.example.net/resource",
11.  "iss": "https://server.example.com/",
12.  "exp": 1419356238,
13.  "iat": 1419350238,
14.  "extension_field": "twenty-seven"
15. }
```

Listing 5: OAuth 2.0 Introspektion [OA15]

Der *Client* im OAuth Rollenkonzept ist eine Anwendung (Software), die geschützte Ressourcen im Namen des Benutzers aufruft. Dies kann z.B. eine Kalender-Anwendung sein, die man berechtigt, auf die *Google Calendar API* zuzugreifen. Dafür wird der Token, ausgestellt vom Authorization Server, genutzt, um HTTP Aufrufe zum Ressource Server durchzuführen. Als Typen von Clients wird im RFC 6749 [Th18a] zwischen zwei Arten von Clients unterschieden. Bei *confidential* Clients handelt es sich um Anwendungen, die in geschützten Umgebungen betrieben werden (z.B. auf einem Server). Von *public* Clients wird gesprochen, wenn diese keine vertraulichen Informationen beinhalten können, wie z.B. die Installationsdateien von Anwendungen (Apps) auf mobilen Endgeräten oder Browser Anwendungen.

Neben der Definition der verschiedenen Rollen werden durch das Protokoll Abläufe für die Autorisierung definiert. Je nach Einsatzzweck und Anwendungstyp werden

unterschiedliche Abläufe eingesetzt, wie den *Authorization Code Flow* für native Anwendungen oder den *Implicit Code Flow* für Browser-basierte Anwendungen.

Nachfolgendend (siehe Abbildung 6) ist der *Authorization Code Flow* dargestellt, da dieser für native Anwendungen von Relevanz ist. Im Ersten Schritt (A) ruft der Client den User-Agent für die Authentifizierung beim Authorization Server auf. Im folgenden Schritt (B) erfolgt die Anmeldung des Benutzers mit der gewählten Methode des Servers (nicht Bestandteil des OAuth 2.0 Frameworks, d.h. es kann z.B. Benutzername + Passwort oder ein anderes Verfahren sein). Nach erfolgreicher Authentifizierung stellt der Authorization Server in Schritt (B) einen Authorization Code aus und leitet den User-Agent anhand der Redirection URI wieder zurück an die Client Anwendung, welche in Schritt (D) bzw. (E) diesen gegen den Access Token austauscht. Nach Durchführung dieses *Authorization Code Flow* kann der Client diesen Access Token für den Zugriff auf die geschützte Anwendung über den Ressource Server verwenden.

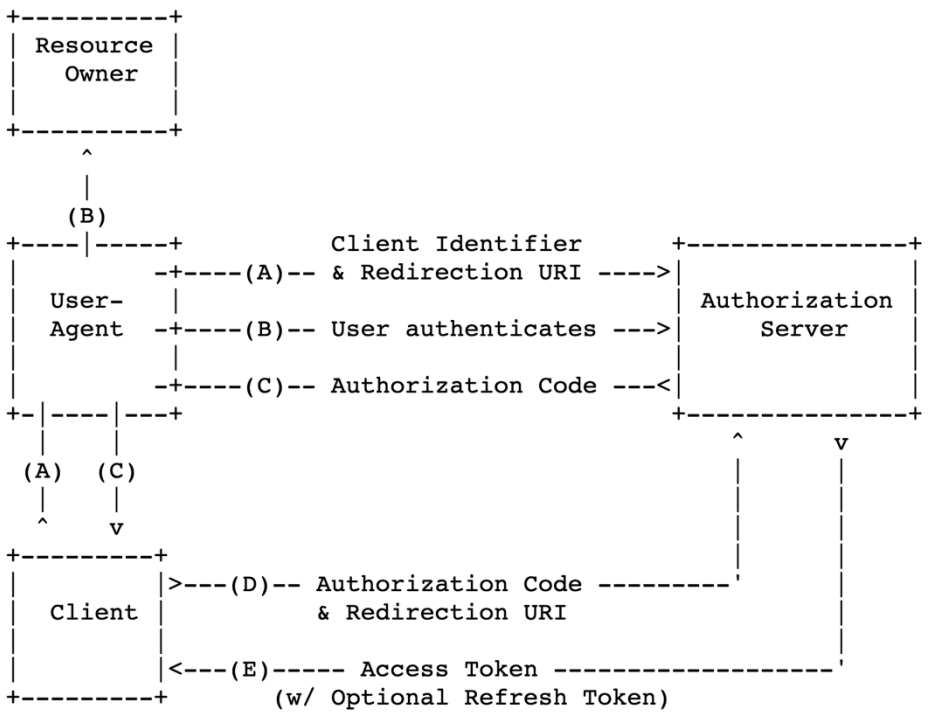

Abbildung 6: OAuth 2.0 Authorization Code Flow [Th18a]

Im Gegensatz zu SAML ist nicht im Detail definiert, wie der Austausch der Informationen zwischen den Akteuren in OAuth 2.0 stattfindet. Daher wird OAuth als Framework und nicht als Protokoll definiert. In den nachfolgenden Abschnitten werden Erweiterungen bzw. Einsatzszenarien beschrieben, wie OAuth 2.0 und das Konzept der delegierten Autorisierung in verschiedenen Anwendungsfällen eingesetzt werden kann. Diese Erweiterungen erfassen ebenfalls den Einsatz von Bearer Tokens, die nicht näher in OAuth spezifiziert sind.

2.1.2.2 OpenID Connect Überblick

In diesem Abschnitt wird ein kurzer Überblick über das Protokoll OpenID Connect dargestellt. Der Einsatzbereich des Protokolls ist für dezentrale Authentifizierungsprozesse nicht nur im Desktop-Bereich, sondern vor allem auch für den einfachen Einsatz im mobilen Bereich konzipiert. OpenID liegt aktuell in der dritten Version vor und stellt eine Schicht oberhalb von OAuth 2.0 dar. In diesem Zusammenspiel (siehe Abbildung 7) wird OpenID für die Anmeldung (Authentifizierung) und OAuth 2.0 für die Berechtigung des Zugriffs auf APIs (Autorisierung) verwendet. Durch dieses Zusammenspiel soll OpenID von den grundlegenden Eigenschaften von OAuth 2.0 profitieren – zum Beispiel Einfachheit, Sicherheit und Unterstützung von diversen Anwendungsszenarien. Für den Austausch der Daten zur Identifizierung, Authentifizierung und Autorisierung zwischen den beteiligten Systemen und Akteuren setzt OpenID Connect auf die Standards REST bzw. JOSE wie z.B. JSON Web Tokens [JS15] auf.

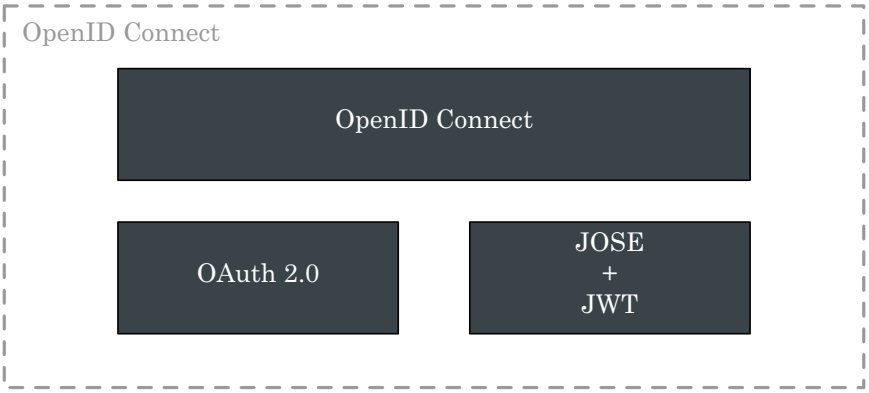

Abbildung 7: OpenID Connect und OAuth 2.0

OpenID Connect soll darüber hinaus einen Standard für

1. Single Sign-On (SSO) und
2. Federation

darstellen.

Die erste Version – entworfen von Brad Fitzpatrick – wurde bereits 2005 verabschiedet, dieser erlangte aber keine große Verbreitung. Kurz nach der Gründung der OpenID Foundation im Jahr 2007 wurde die finale Version von OpenID 2.0 veröffentlicht. OpenID Connect ist die dritte Version des Standards, welcher im Februar 2014 durch die OpenID Foundation publiziert wurde. Ein Jahr später wurde ein Zertifizierungsprogramm im April 2015 gestartet. Mittlerweile verwenden eine Vielzahl an großen Internetfirmen (Google, Microsoft, Yahoo, Facebook, Apple) diesen Standard zur Authentifizierung. Das Protokoll wird aktuell von der OpenID Foundation verwaltet. Diese wurde im Juni 2007 als gemeinnützige (non-profit) Organisation gegründet, um die Weiterentwicklung des Protokolls und die Koordinierung der Community zu steuern.

OpenID Connect – Überblick und Funktionsweise

OpenID Connect stellt ein – vergleichbar mit SAML – Verfahren dar, damit sich Benutzer bei einer zentralen Stelle anmelden (OpenID Provider) und eine Vielzahl

an Anwendungen (Ressource Providern), ohne weitere Authentifizierung, nutzen zu können. Mit der letzten Version OpenID Connect 1.0 [Op14] wird das Authentifizierungsprotokoll mit OAuth 2.0 [Th18a] für die Autorisierung und weiteren bestehenden Standards verknüpft – siehe nachfolgende Abbildung 8:

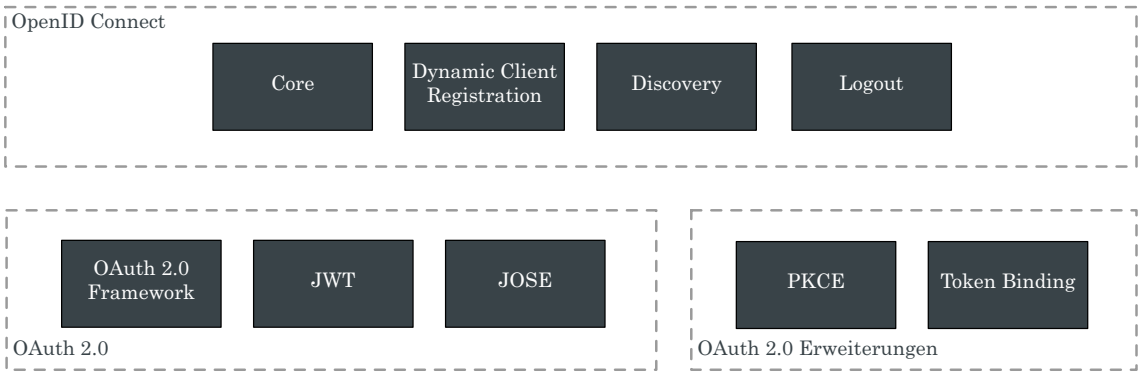

Abbildung 8: OpenID Connect Protokoll Familie (in Anlehnung an [SM18c])

- OpenID Connect Core – definiert die Kernelemente des Protokolls für die Authentifizierung auf Basis von OAuth 2.0
- OpenID Connect Discovery – definiert, wie Clients dynamisch Informationen von OpenID Provider beziehen
- OpenID Connect Dynamic Client Registration – definiert die Registrierung von Clients bei OpenID Providern
- OpenID Connect Session Management – definiert ein einheitliches Session Management für ein Logout des Benutzers
- OpenID Connect Form Post Response Mode – definiert, wie eine Autorisierungs-Antwort in eine http post Nachricht an den Client übermittelt werden kann

Aufbauend auf dieser Protokoll-Familie stehen noch weitere Spezifikationen (z.B. für den Einsatz im Finanzsektor) zur Verfügung [Sp20a].

Im Gegensatz zu älteren Standards wie SAML (Security Assertion Markup Language) ist OpenID Connect sowohl für den Einsatz von Web-Clients als auch im mobilen Bereich konzipiert. Der Austausch erfolgt dabei über Tokens (z.B. Access

und ID Tokens), die im JSON Web Tokens (JWT) [JS15] ausgetauscht werden. Damit soll OpenID Connect leicht auf verschiedenen Plattformen integriert werden können und eine Verarbeitung der Tokens einfach sein [Sr14].

Der ID Token [Open00] repräsentiert das Konzept eines Identitätsdokuments im digitalen Format (vergleichbar der SAML Assertion siehe Abschnitt 2.1.1.1), bestätigt (signiert) durch den OpenID Connect Provider (OP) und beinhaltet folgende Elemente (Claims):

- Bestätigung der Benutzer-Identität *(sub),*
- Ausstellende Stelle *(iss),*
- Empfänger (Client) für diesen Token *(aud),*
- Zufallswert *(nonce),*
- Authentifizierungszeit (*auth_time)* und Authentifizierungsstärke *(acr)* des Benutzers,
- Ausstellungsdatum *(iat)* und
- Ablaufdatum *(exp).*

Nachfolgend ist ein Inhalt eines ID Tokens in dekodierter Darstellung (Listing 6) bzw. in enkodierter Darstellung (Listing 7) im JSON Web Token Format abgebildet.

```
1. {
2.    "iss"       : "https://server.example.com",
3.    "sub"       : "24400320",
4.    "aud"       : "s6Bhdrkqt3",
5.    "nonce"     : "n-0S6_WzA2Mj",
6.    "exp"       : 1311281970,
7.    "iat"       : 1311280970,
8.    "auth_time" : 1311280969,
9.    "acr"       : "urn:mace:incommon:iap:silver"
10. }
```

Listing 6: Beispiel OpenID ID Token (dekodiert) [Op14]

```
1. eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazcifQ.ewogImlzcyI6
2. ICJodHRwOi8vc2VydmVyLmV4YW1wbGUuY29tIiwKICJzdWIiOiAiMjQ
3. 4Mjg5NzYxMDAxIiwKICJhdWQiOiAiczZCaGRSa3F0MyIsCiAibm9uY2
4. UiOiAibi0wUzZfV3pBMk1qIiwKICJleHAiOiAxMzExMjgxOTcwLAogI
5. mlhdCI6IDEzMTEyODA5NzAKfQ.ggW8hZ1EuVLuxNuuIJKX_V8a_OMXz
6. R0EHR9R6jgdqrOOF4daGU96Sr_P6qJp6IcmD3HP99Obi1PRs-cwh3LO
7. -p146waJ8IhehcwL7F09JdijmBqkvPeB2T9CJNqeGpe-gccMg4vfKjk
8. M8FcGvnzZUN4_KSP0aAp1tOJ1zZwgjxqGByKHiOtX7TpdQyHE5lcMiK
9. PXfEIQILVq0pc_E2DzL7emopWoaoZTF_m0_N0YzFC6g6EJbOEoRoSK5
10. hoDalrcvRYLSrQAZZKflyuVCyixEoV9GfNQC3_osjzw2PAithfubEEB
11. LuVVk4XUVrWOLrLl0nx7RkKU8NXNHq-rvKMzqg
```

Listing 7: Beispiel OpenID ID Token (enkodiert) [Op14]

Der erste Teil der Nachricht (Zeile eins des Listing 7) gibt die Meta-Informationen an (Algorithmus der Verschlüsselung), im zweiten Teil (Zeile zwei bis fünf des Listing 7) wird der Inhalt (ID-Token) kodiert und der letzte Teil (Zeile sechs bis zwölf des Listing 7) beinhaltet die Signatur des JWT. Um diesen ID Token bzw. Access Token vom Identity Provider abzufragen, definiert das Protokoll drei Abläufe (flows) [Open00]:

1. Authorization Code Flow
    A. Tokens werden über getrennte Back-Channels zurückgesendet
    B. der Client wird authentifiziert
    C. Flow kann in mobilen Apps und Web-Anwendungen eingesetzt werden
2. Implicit Flow
    A. Tokens werden direkt zurückgesendet
    B. der Client wird nicht authentifiziert
    C. Flow dient vor allem für JavaScript Anwendungen ohne Backend
3. Hybrid Flow – Kombination beider oben dargestellten Abläufe.

Nachfolgend (siehe Abbildung 9) wird ein minimales Beispiel anhand des *Authorization Code Flow* dargestellt [Op14]. Der Ablauf kann in zwei Stufen unterteilt werden:

- Schritt 1.A bis 1.C: Autorisierung durchführen
- Schritt 2: Access Token abrufen

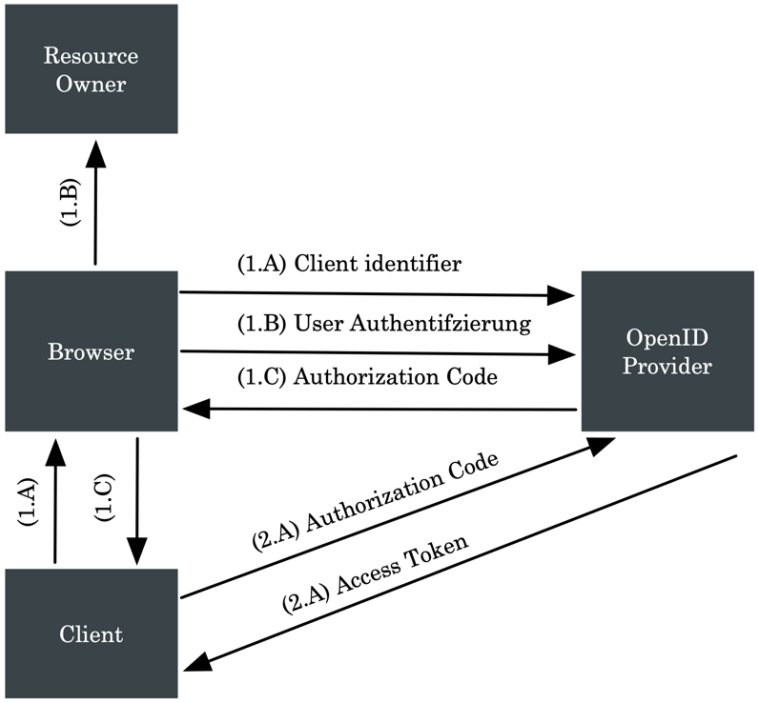

Abbildung 9: Authorization Code Flow (in Anlehnung [Op14])

Im **ersten Schritt** (Listing 8) wird die Initialisierung der Authentifizierung durchgeführt, welche mit dem *Authorization Code* bestätigt wird.

1. Authentifizierungs-Request wird an den OpenID Provider gesendet.
2. Durchführung der Authentifizierung.
3. Redirect des Browsers mit dem *Authorization Code.*

```
1. HTTP/1.1 302 Found
2. Location: https://client.example.com/cb?
```

```
3.            code=SplxlOBeZQUYbYS6WxSbIA
4.            &state=ag0ifjsldkj
```

Listing 8: Beispiel OpenID Authorization Code Flow Schritt 1 [Op14]

Im **zweiten Schritt** (Listing 9) wird ein Token Request an den OpenID Provider gesendet. Dieser sendet anschließend der Client Anwendung den jeweiligen Token.

1. Client authentifiziert sich mit dem *Authorization Code* gegenüber den OpenID Provider.
2. OP sendet den Token (siehe oben) an den Client.

```
1. POST /token HTTP/1.1
2. Host: server.example.com
3. Content-Type: application/x-www-form-urlencoded
4. Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
5.
6. grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA
7.   &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
```

Listing 9: Beispiel OpenID Authorization Code Flow Schritt 2 [Op14]

2.1.2.3 OpenID Connect – Native Mobile Apps

Der RFC *OAuth 2.0 for Native Apps* [OA17] definiert eine Verfahrensanleitung, wie in einer mobilen Anwendung der Authentifizierungsablauf durchgeführt werden soll. Bei mobilen Anwendungen (Apps) handelt es sich um einen öffentlichen und nicht vertrauenswürdigen (untrusted) Client [Pa17]. Um dem Benutzer/der Benutzerin ausreichend Sicherheit garantieren zu können, ist daher empfohlen, den Anmeldevorgang nicht direkt in der App durchzuführen, sondern den Benutzer/die Benutzerin über dessen/deren Browser zum Anmeldevorgang zu leiten. Nach erfolgreicher Durchführung der Anmeldung wird der Benutzer/die Benutzerin über Redirects wieder zur App geleitet. Der Ablauf der Anmeldung ist in der Abbildung 10 dargestellt [Sr20a]:

1. Der Benutzer/die Benutzerin startet die Anmeldung (*Authorization Request*) und wird in den Browser übergeleitet.
2. Im Browser führt der Benutzer/die Benutzerin die Anmeldung durch und nach erfolgreicher Durchführung,
3. wird über den Browser der *Authorization Code* per Redirect an die Client App übergeben.
4. Diese nimmt den *Authorization Code* entgegen und
5. löst diesen beim Token Endpoint gegen
6. den *Access Token* bzw. *Refresh Token* ein.

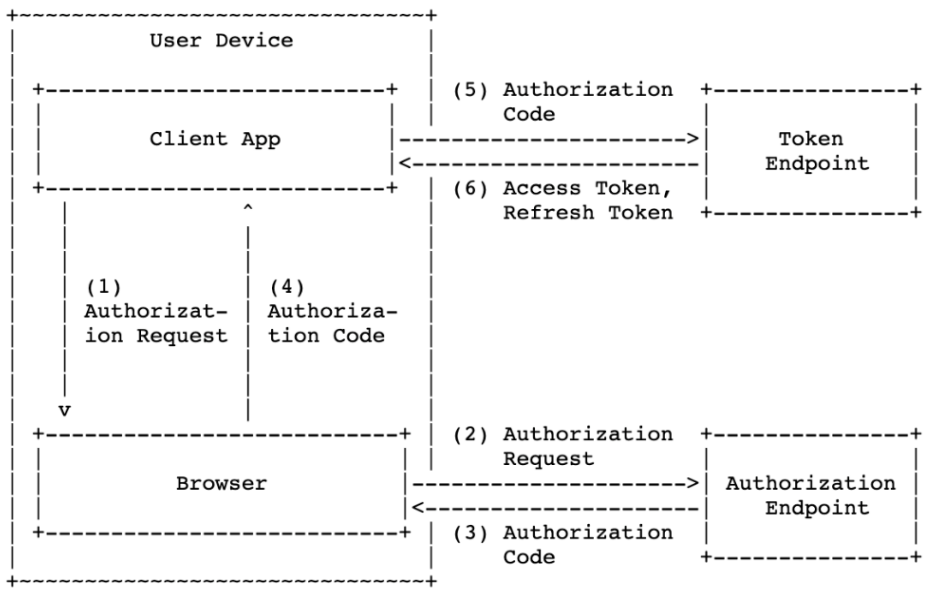

Abbildung 10: OpenID Connect – Native Mobile Apps [OA17]

Nach dem Abrufen des *Access Token* (Schritt 6) kann die App die geforderte API aufrufen und den Token einlösen.

Im Gegensatz zur direkten Implementierung der Abläufe in der App oder über eine WebView ist damit sichergestellt, dass die App selbst niemals in die Verwendung der Credentials des Benutzers kommt. Darüber hinaus ist die Verwendung des *client_secret* gemäß [Op14] nicht zu empfehlen, da dieses nicht geheim gehalten werden kann.

2.1.2.4 OpenID Connect – Proof Key for Code Exchange (PKCE)

Die Erweiterung des Anmeldevorganges gemäß dem RFC 7636 [Pr15] soll verhindern, dass im Zuge des Anmeldevorganges eine nicht vertrauenswürdige bzw. bösartige App den *Authorization Code* abfängt (siehe Schritt 4 in Abbildung 11) und sich damit den *Access Token* vom Token Endpunkt abholt. Durch die Empfehlung des RFC 8252 [OA17] für native Apps (siehe vorheriger Abschnitt 2.1.1.3) wird für die Durchführung des Anmeldevorgangs der Browser des Benutzers/der Benutzerin gestartet und nach erfolgreicher Durchführung der Authentifizierung (über einen Redirect) zurück zur nativen App umgeleitet. Dabei ist es möglich, dass neben der gewünschten App, eine bösartige auf diesen Redirect reagiert und anschließend in den Besitz des *Authorization Code* gelangt.

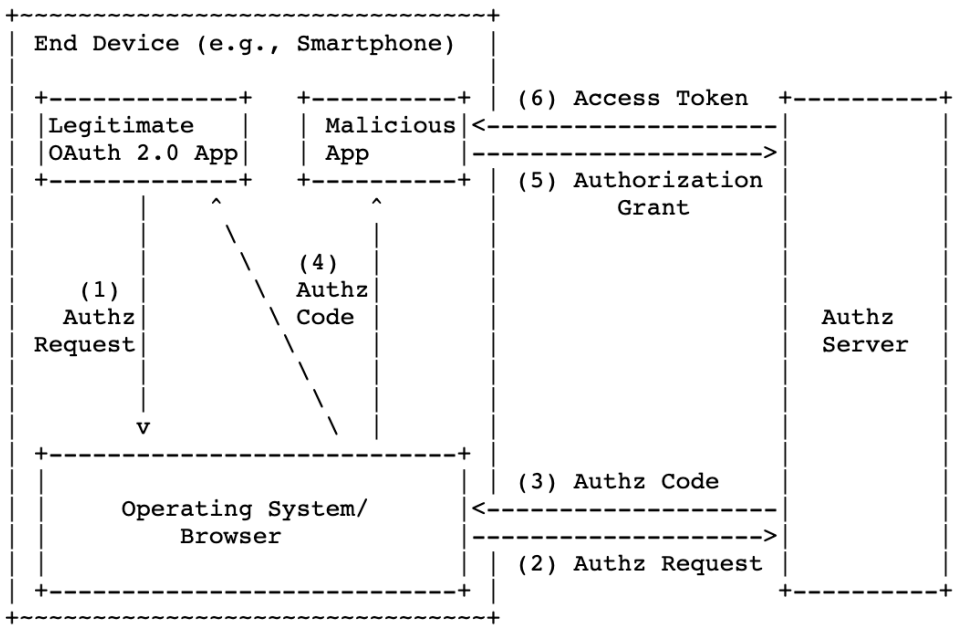

Abbildung 11: OpenID Connect Authorization Code abfangen [OA17]

Aufgrund des Umstandes bei öffentlichen Apps kein *client_secret* verwenden zu können, wird mit PKCE der Authentifizierungsvorgang um eine Challenge (siehe Abbildung 12) erweitert, die nur der vertrauenswürdigen App und dem Authorization Server bekannt ist.

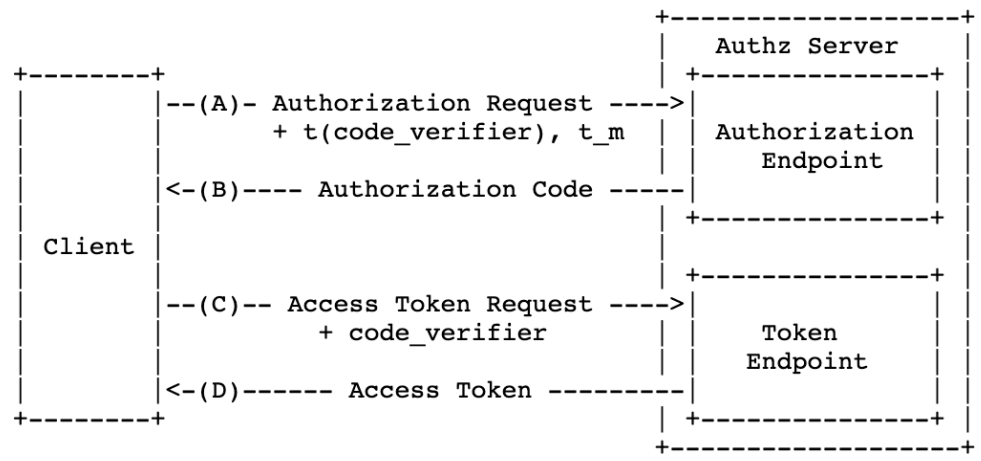

Abbildung 12: PKCE OpenID Connect Erweiterung [OA17]

Damit ein Angriff verhindert werden kann, wird mit dem *Authorization Request* ein mittels Hash verschleierter Zufallswert *(code_verifier)* an die *Authorization Server* gesendet (A). Zu diesem Zeitpunkt ist dieser Zufallswert nur dem Client bekannt und dem Authorization Server der Hash über diesen Zufallswert. Zum Zeitpunkt der Ausstellung der Authorization Code (B) bindet der Authorization Server diesen an den Hash über den Zufallswert. Im Zuge des Access Token Request (C) zum Einlösen des Authorization Code gegen den Access Token, übermittelt der Client die Zufallszahl *(code_verifier)* im Klartext an den Server. Dieser kann bei der Prüfung des Authorization Code, welcher an den Hash über die Zufallszahl gebunden ist, prüfen, ob der Client im Besitz der Zufallszahl *(code_verifier)* ist und liefert bei erfolgreicher Verifikation (D) den Access Token zurück. Mit diesem Verfahren wird ermöglicht, dass ein Einlösen des Authorization Code (siehe Abbildung 12) nur von der App durchgeführt werden kann, die den Authentifizierungsvorgang gestartet hat.

2.1.2.5 OpenID Connect – Token Binding

Nach erfolgreicher Authentifizierung des Benutzers/der Benutzerin kann jeder der in den Besitz des *Access* bzw. *Refresh Tokens* gelangt auf geschützte Ressourcen zugreifen. Damit können Dritte unbemerkt auf Ressourcen zugreifen und dem

Ressource Server ist es nicht möglich, dies zu unterbinden. Mit der Erweiterung des RFC 8741 [Th18b] werden die Bearer Tokens kryptographisch an die TLS Verbindung gebunden, um damit ein Entwenden der Tokens zu verhindern. Der Ablauf [Sr20c] lässt sich anschaulich in drei Phasen (siehe Abbildung 13) unterteilen. In der ersten *Negotiation Phase* werden die Schlüsselparameter zwischen Client und Server für das Token Binding Protokoll vereinbart. Darauffolgend generiert der Client für jeden Server oder jede Domain ein Schlüsselpaar in einer sicheren Umgebung, sodass dieses nicht exportiert werden kann. Idealerweise werden die notwendigen kryptografischen Schlüssel in einem TEE (Trusted Execution Environment), wie einem TPM Chip oder einem anderen dedizierten Security Chip auf mobilen Endgeräten, generiert. Die ausgestellten Tokens werden an das generierte Schlüsselpaar gebunden und der Client muss in der *Proof of Possesion Phase* nachweisen, noch im Besitz des privaten Schüssels zu sein, um die Tokens (z.B. Access Token) verwenden zu können.

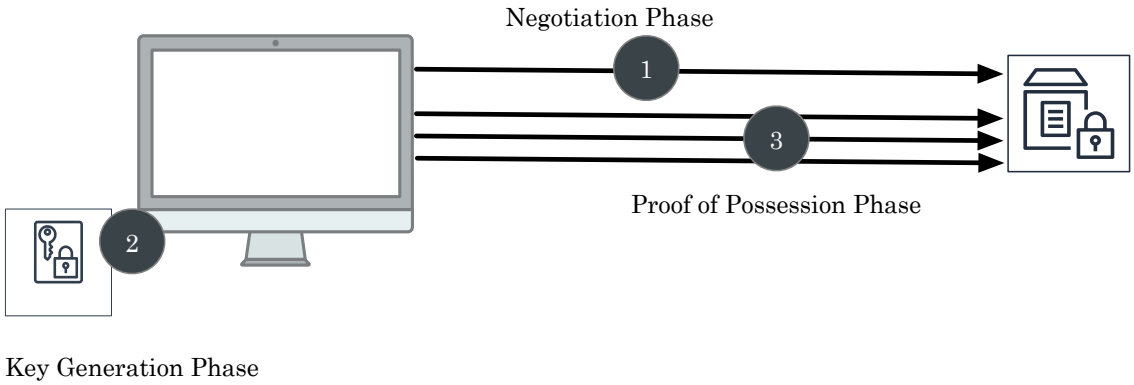

Abbildung 13: OAuth 2.0 Token Binding (in Anlehnung an [Sr20c])

Mit der Erweiterung des Transport Layer Security (TLS) Protokolls gemäß RFC 8472 [Tr18] wird das Token Binding ermöglicht. Beide Seiten – sowohl der Client/Browser, als auch der Server müssen diese Erweiterung des TLS Protokolls unterstützen, was den Einsatz aufgrund der geringen Verbreitung einschränkt.

2.1.2.6 OAuth 2.0 Mutual TLS Authentication

Der RFC *OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens* [OA20] definiert, wie Access und Refresh Tokens an ein TLS Client Zertifikat gebunden werden bzw. ein OAuth 2.0 Client sich beim Authorization Server authentifizieren können.

In der nachfolgenden Abbildung 14 authentifiziert sich der Client (Schritt A) gegenüber dem Authorization Server mit dem TLS Client Zertifikat beim Authorization Server (Token Endpunkt). Anschließend wird der Access und Refresh Token an dieses TLS Client Zertifikat gebunden und im Schritt B, im Zuge des Zugriffs auf eine geschützte Ressource, kann der Ressource Server prüfen, ob der Access Token für dieses TLS Client Zertifikat ausgestellt wurde.

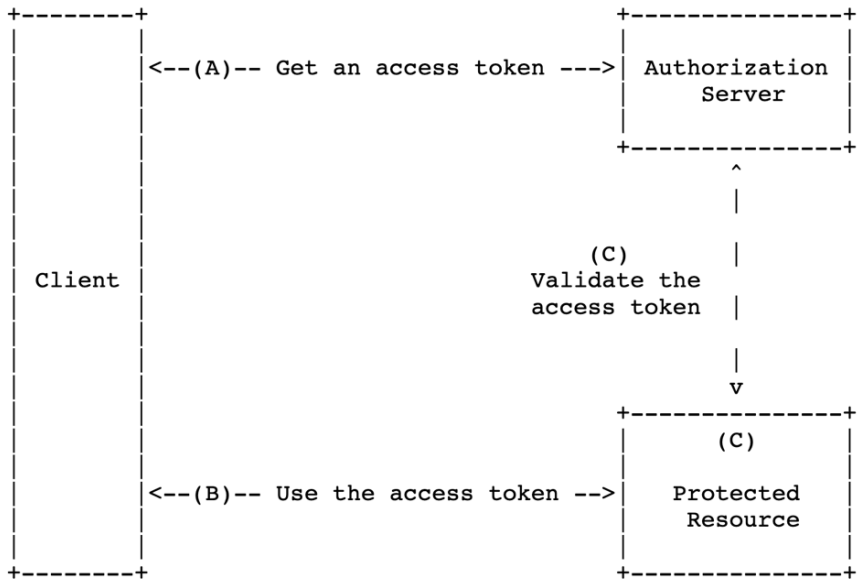

Abbildung 14: OAuth 2.0 Mutual TLS Authentication [OA20]

Im Falle von öffentlichen Clients wird die Authentifizierung des Clients nicht mit Hilfe des TLS Clients Zertifikats durchgeführt. Jedoch verwendet der Authorization Server das TLS Client Zertifikat bei der Bindung der Access und Refresh Tokens an das TLS Client Zertifikat. Im Unterschied zum Token Binding Protokoll (siehe Abschnitt 2.1.2.5) muss beim RFC 8705 der TLS Handshake nicht erweitert werden,

sondern es werden zwischen dem Client/Browser und dem Server TLS-Zertifikate verwendet. Dieses TLS Zertifikat (siehe Zeile 8 in Listing 10) wird dann z.B. im Access Token über einen Hash über das Zertifikat referenziert, sodass dies beim Aufruf einer Ressource überprüft werden kann.

```
1. {
2.    "active": true,
3.    "iss": "https://server.example.com",
4.    "sub": "ty.webb@example.com",
5.    "exp": 1493726400,
6.    "nbf": 1493722800,
7.    "cnf":{
8.    "x5t#S256": "bwcK0esc3ACC3DB2Y5_lESsXE8o9ltc05O89jdN-dg2"
9.    }
10. }
```

Listing 10: OAuth 2.0 Mutual TLS Authentication [OA20]

2.1.3 Bürgerkarte, Handy-Signatur und E-ID

Die Bürgerkarte kann in Form einer Signaturkarte (z.B. E-Card) oder als Handy-Signatur genutzt werden und stellt einen technologieneutralen Begriff für einen sicheren Token dar [TH19]. Auf der Bürgerkarte werden die kryptographischen Schlüssel in einem sicheren und zertifizierten Datenspeicher abgelegt und können nur durch Eingabe eines Passworts verwenden werden. Dieses Konzept, wie es von einer physischen Karte (Chip-Karte) bekannt ist, wurde übernommen und als Handy-Signatur verwendet.

2.1.3.1 Bürgerkarte und Handy-Signatur

Um die Verbreitung der Bürgerkarte zu erhöhen, wurde 2009 das Konzept der Handy-Signatur umgesetzt. Dabei werden die Schlüssel und Berechnungen nicht auf einer lokalen Bürgerkarte (Signaturkarte) abgelegt bzw. durchgeführt, sondern auf serverbasierten Lösungen.

Mit der ersten Version der Bürgerkarte (Chipkarte und Handy-Signatur) kann eine nach österreichischem Recht [ZTL11] konforme digitale Unterschrift erstellt werden, entweder lokal durch die Signaturkarte oder über die Handy-Signatur. Dieses Verfahren wurde genutzt, um damit eine Anmeldung durchführen zu können, siehe Abbildung 15 am Beispiel der Handy-Signatur. Bei einer Anmeldung bei einem Service Provider wird somit mit einer digitalen Unterschrift unterzeichnet.

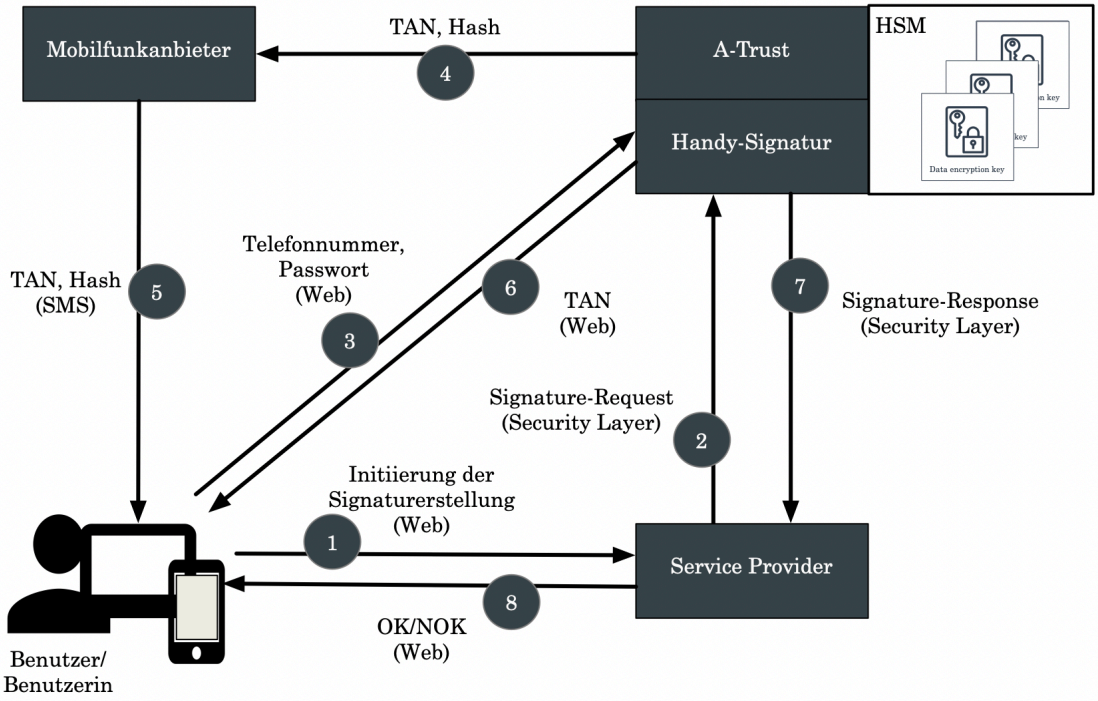

Abbildung 15: Authentifizierung mittels Handy-Signatur (in Anlehnung an [ZTL11])

Der Ablauf gestaltet sich hierbei folgend:

1. Die Anmeldung wird bei der Anwendung (Service Provider) ausgelöst.
2. Dieser löst eine Signatur-Erstellung über das Security Layer Protokoll aus.
3. Der Benutzer/die Benutzerin übermittelt über ein Web-Formular seine/ihre Telefonnummer und Passwort.
4. Die Handy-Signatur übermittelt einen TAN und den Hash der Transaktion über

5. den Mobilfunkanbieter über einen zweiten Kanal an den Benutzer/die Benutzerin. Anfangs wurde zur Übertragung der TAN und des Hashs eine SMS übermittelt. In der aktuellen Version der Handy-Signatur oder Digitalen Amt App erfolgt dies über eine Push Notifikation der Handy-Signatur an den Nutzer.
6. Dieser gibt den zweiten Faktor in das Web-Formular ein (SMS) oder bestätigt die Transaktion direkt in der Handy-Signatur bzw. Digitales Amt App und löst damit die
7. Erstellung der persönlichen Signatur aus, welche an den Service Provider zurückgeliefert wird.
8. Nach erfolgreicher Erstellung und Prüfung der Signatur (inkl. Durchführung einer Anmeldung) ist der Benutzer/die Benutzerin an der gewünschten Anwendung angemeldet.

Damit dieser Ablauf [TH19] durchgeführt werden kann, werden für jeden Benutzer serverbasiert die notwendigen Schlüssel und Zertifikate sicher aufbewahrt (zertifiziertes Hardware Security Modul, beim Betreiber A-Trust). Nur durch das Zusammenspiel von Wissen (Passwort) und Besitz (Telefon) wird ein Zugriff auf den Schlüssel freigeschaltet und die Anmeldeinformationen signiert. Der dargestellte Ablauf lässt sich auf die kartenbasierte Lösung der Bügerkarte übertragen. Hierbei erfolgt die Erstellung der Signatur nicht durch eine lokale Sichere Signaturerstellungseinheit (SSEE). Beide Lösungen wurden für den Einsatz der sicheren Authentifizierung von Webseiten konzipiert. Für die Identitätsfeststellung wurde die österreichische Lösung MOA-ID (Modul für Online Applikationen – Identität) als IDP (Identitiy Provider) verwendet und als Authentifizierungsstandards kommen SAML und OpenID Connect zum Einsatz. Das Verfahren zur Authentifizierung kann in native Anwendungen auf mobilen Endgeräten nicht integriert werden und somit entstand das Verfahren zum E-ID (Elektronischer Identitätsnachweis) - siehe nächster Abschnitt 2.1.3.2.

Um diesen Ablauf der Authentifizierung nicht von jedem Service Provider implementieren zu müssen, wird vom EGIZ (E-Government Innovationszentrum) eine Software-Komponente (MOA-ID) – siehe Abbildung 16 – zur Verfügung gestellt, welche je nach Anwendungsfall als SAML Identity Provider oder OpenID Connect Provider eingesetzt werden kann. [LE14].

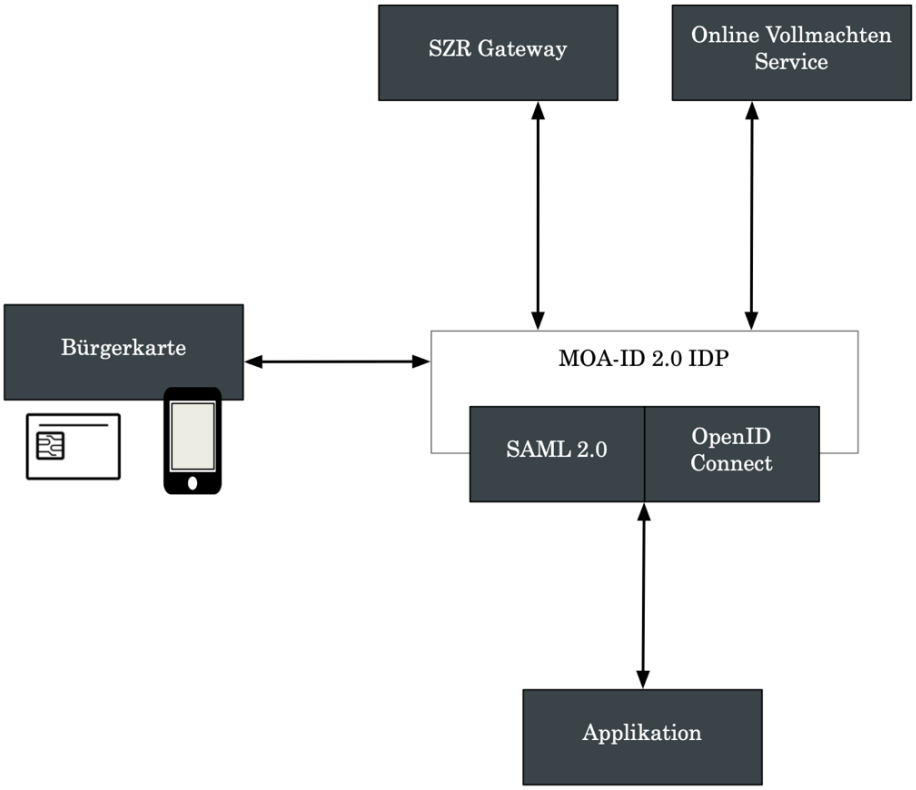

Abbildung 16: MOA-ID

Der dargestellte Ablauf einer Anmeldung kann für die Nutzung in einer mobilen Anwendung nicht durchgeführt werden [TH19], sondern erfordert eine Erweiterung der aktuellen Konzepte und Umsetzungen, wie der nächste Abschnitt 2.1.3.2 zeigt.

2.1.3.2 Elektronischer Identitätsnachweis (E-ID)

Mit der Erweiterung der österreichischen Bürgerkartenlösung zum E-ID (elektronischer Identitätsnachweis) wird unter anderem ein Einsatz im Bereich der

mobilen Anwendungen ermöglicht. Neben dieser Erweiterung des Einsatzgebietes ist eine weitere wesentliche Änderung zum aktuellen Konzept, dass die dezentralen Identity Provider (bzw. Authorization Server) entfallen und eine zentrale Lösung realisiert wird. Alle Service Provider bzw. Ressource Server müssen sich in einem zentralen Applikationsregister registrieren und können nach erfolgreicher Akkreditierung das E-ID System nutzen. Die Attribute (SAML) bzw. Claims (OpenID Connect) sind mit der neuen Architektur in nachgelagerten Registern der Behörden gespeichert und werden nach der Anmeldung an den Service Provider / Ressource Server (Applikation) bzw. Client übermittelt. Nachfolgend – siehe Abbildung 17 – ist eine abstrakte Darstellung der E-ID Gesamtarchitektur [AR20] visualisiert.

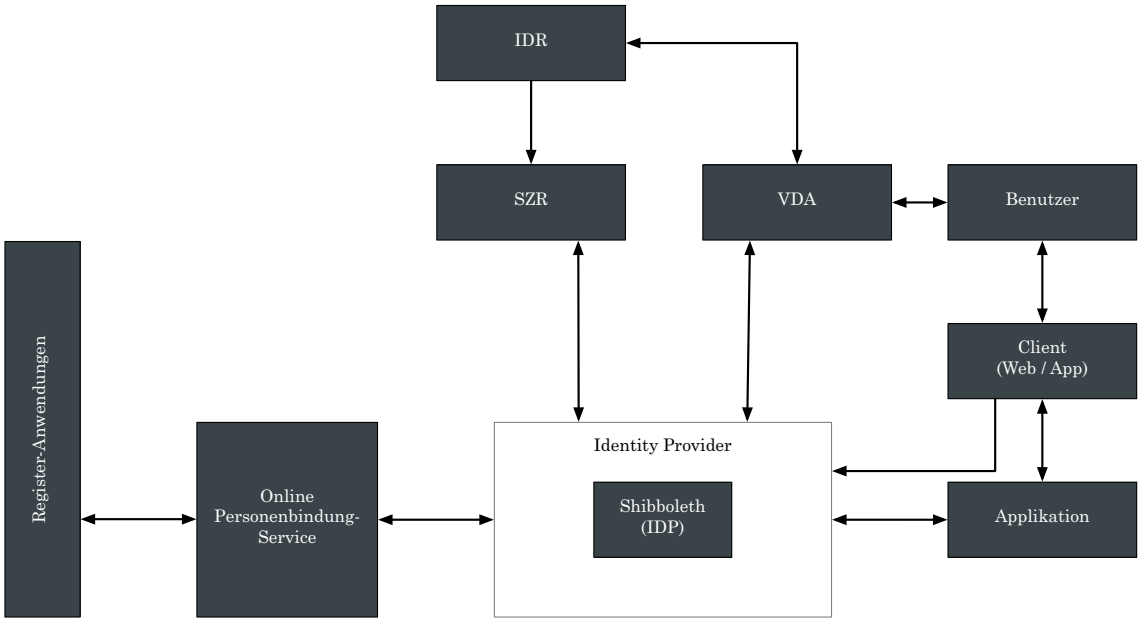

Abbildung 17: E-ID Gesamtarchitektur (in Anlehnung an [AR20])

In Abbildung 18 wird der Ablauf einer OpenID Connect basierten Anmeldung im mobilen Bereich dargestellt (Anmeldungen über das SAML Protokoll für mobile Anwendungen werden nicht unterstützt). Der Ablauf orientiert sich abstrakt an der Empfehlung für die Authentifizierung von nativen Apps des RFC 8252 (siehe Abschnitt 2.1.2.3). Ein Unterschied dieser beiden Varianten ist allerdings, dass die

Anmeldung nicht über den Browser des mobilen Endgerätes, sondern über die E-ID – Digitales Amt App – durchgeführt wird. [An20a] In den Schritten eins (1) bis drei (3) wird eine Anmeldung des Online Service (Ressource Server) über die E-ID App initiiert. Danach wird die Authentifizierung des Benutzers/der Benutzerin über den zentralen Identity Provider (Shibboleth IDP) und dem VDA (Vertrauensdiensteanbieter) in den Schritten vier (4) bis sechs (6) durchgeführt. Nach erfolgreicher Authentifizierung wird von der E-ID App der Authorization Code in Schritt sieben (7) an die Service App übergeben, welche diesen anschließend gegen den Access bzw. ID Token einlöst.

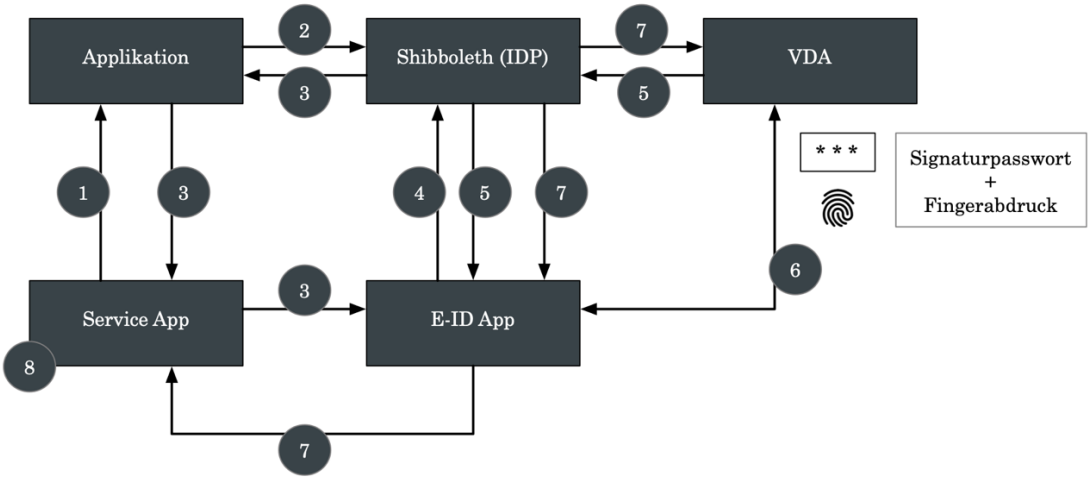

Abbildung 18: E-ID Prozessfluss der Anmeldung (in Anlehnung an [An20a])

Zum Zeitpunkt der Erstellung dieser Arbeit ist das E-ID System gerade im Aufbau und wird mit Ende November/Anfang Dezember 2020 (siehe Abbildung 19) zur Verfügung stehen.

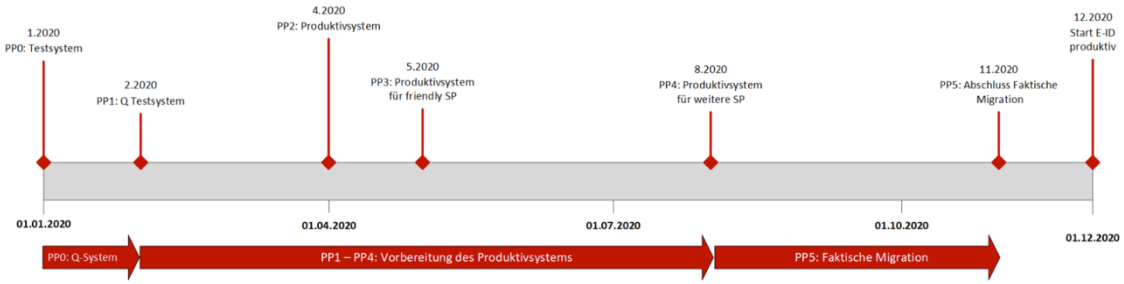

Abbildung 19: E-ID Roadmap [Ee20]

2.1.4 Fast Identity Online (FIDO)

FIDO ist ein Authentifizierungsstandard, der auf Public Key Kryptographie aufbaut und zwischen dem Client und einer Anwendung (Relying Party) eine starke Bindung herstellt (siehe Abbildung 20). Im Gegensatz zu den Federation Protokollen SAML und OpenID Connect ist der grundsätzliche Ansatz jedoch ein anderer, da die Bindung zwischen dem Client und der Relying Party direkt hergestellt wird. Sie lassen sich jedoch, wie in den folgenden Ausführungen noch dargestellt wird, auch mit diesen Federation Protokollen integrieren.

Der Prozess der Nutzung von FIDO [FI17] untergliedert sich in die *Registrierung* und die *Authentifizierung*. Im ersten Schritt wird ein kryptographisches Schlüsselpaar registriert, um dieses dann im nächsten Schritt für die Authentifizierung nutzen zu können. Dafür implementiert der FIDO Client und der FIDO Server das UAF Protocol. Auf Client Seite stellt der FIDO Client die Schnittstelle zum FIDO Authenticator dar. Auf der Gegenseite ist der FIDO Server, der die Registrierung und Nutzung des UAF Protokolls steuert.

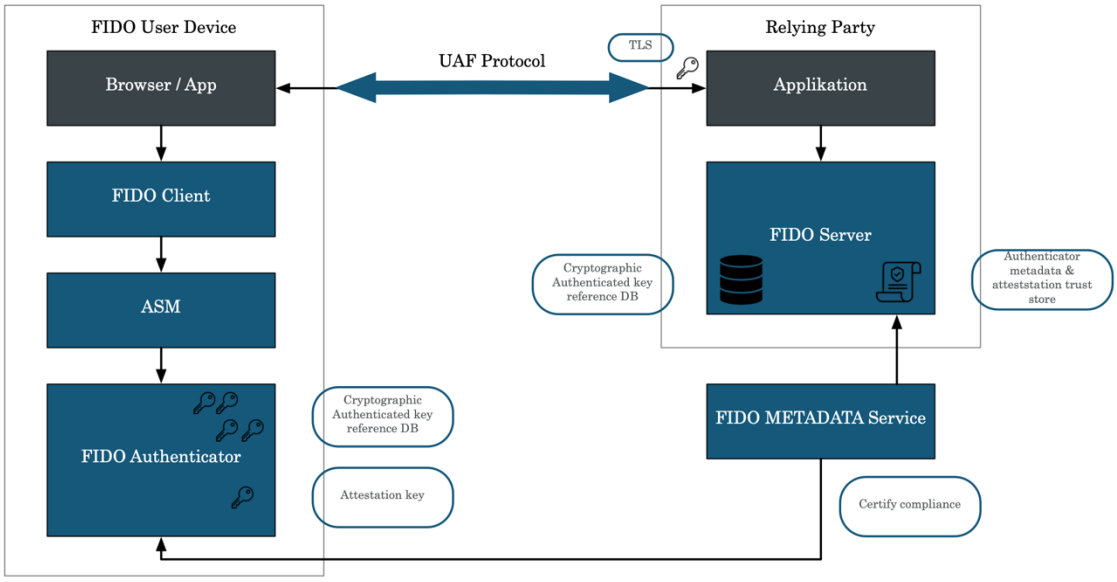

Abbildung 20: FIDO UAF High-Level Architektur (in Anlehnung an [FI17])

Im Zuge der Registrierung wird der Benutzer/die Benutzerin beim Aufruf einer Webseite gefragt, ob er/sie dieses Gerät registrieren möchte. Dabei wird der Benutzer/die Benutzerin nach der Bestätigung seiner/ihrer lokalen Identität gefragt (z.B. über PIN oder Biometrische Merkmale) und es wird lokal ein Schlüsselpaar generiert, welches vom FIDO Authenticator mit dem Attestation key bestätigt wird (siehe Abbildung 21). Für jeden Server wird hierbei ein eigenes Schlüsselpaar generiert, damit der Benutzer/die Benutzerin über mehrere Relying Parties nicht nachverfolgt werden kann.

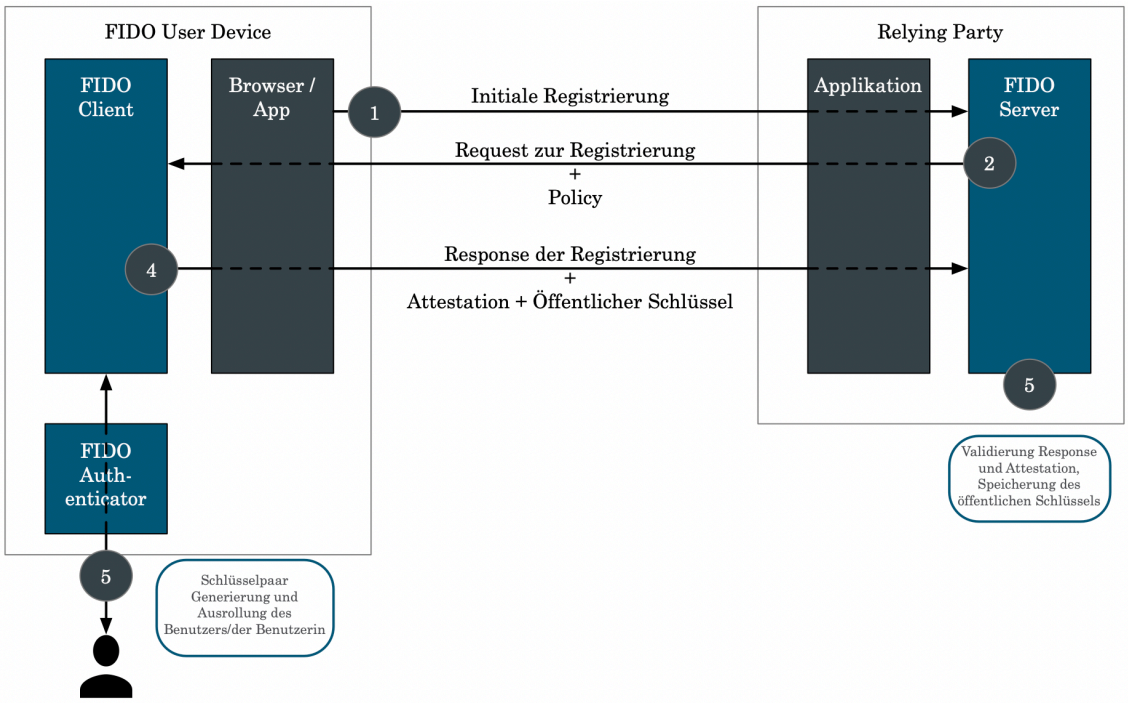

Abbildung 21: FIDO Registrierung (in Anlehnung an [FI17])

Nach der Registrierung des kryptographischen Schlüssels beim FIDO Server kann der Benutzer/die Benutzerin die Authentifizierung gegen eine Relying Party starten (siehe Abbildung 22). Dabei wird die Authentifizierung durch das lokale Schlüsselpaar bestätigt und der FIDO Server kann damit den Besitz prüfen und eine Authentifizierung durchführen.

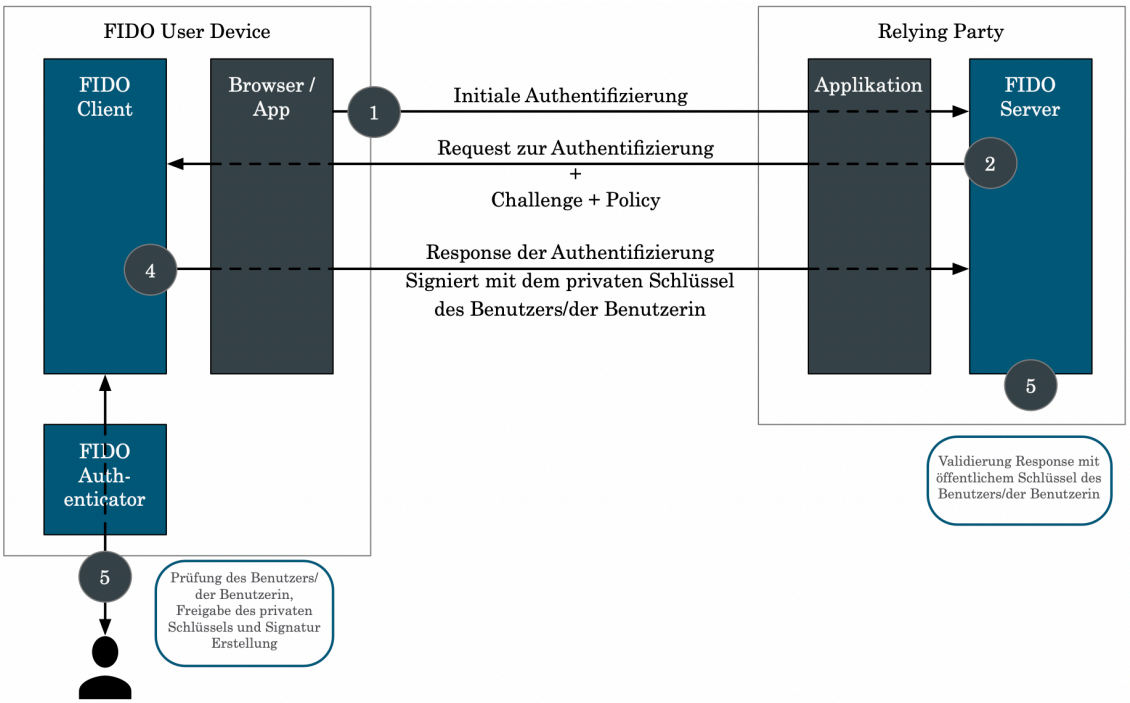

Abbildung 22: FIDO Authentifizierung (in Anlehnung an [FI17])

Durch die direkte Registrierung bei der Anwendung (Relying Party) und nicht bei einem zentralen Identity Provider / OpenID Connect Provider ist der grundsätzliche Ansatz abweichend als bei den genannten Federation Protokollen SAML und OpenID Connect. Hierbei registriert sich der Benutzer/die Benutzerin bei einer zentralen Stelle und nicht bei jeder dezentralen Anwendung. Beide Ansätze lassen sich aber dahingehend kombinieren, dass die Registrierung bei einer zentralen Instanz der Authentication Authority erfolgt (siehe Abbildung 23). Die Authentication Authority stellt hierbei den Identity Provider oder den OpenID Connect Provider dar. Die auf FIDO basierte Registrierung und Authentifizierung erfolgt damit zwischen dem Client und der Authentication Authority, welche nach erfolgreicher Anmeldung eine Bestätigung (Assertion) ausstellt, und diese wird dann für den Aufruf an den Application Provider (Service Provider bzw. Ressource Server) genutzt.

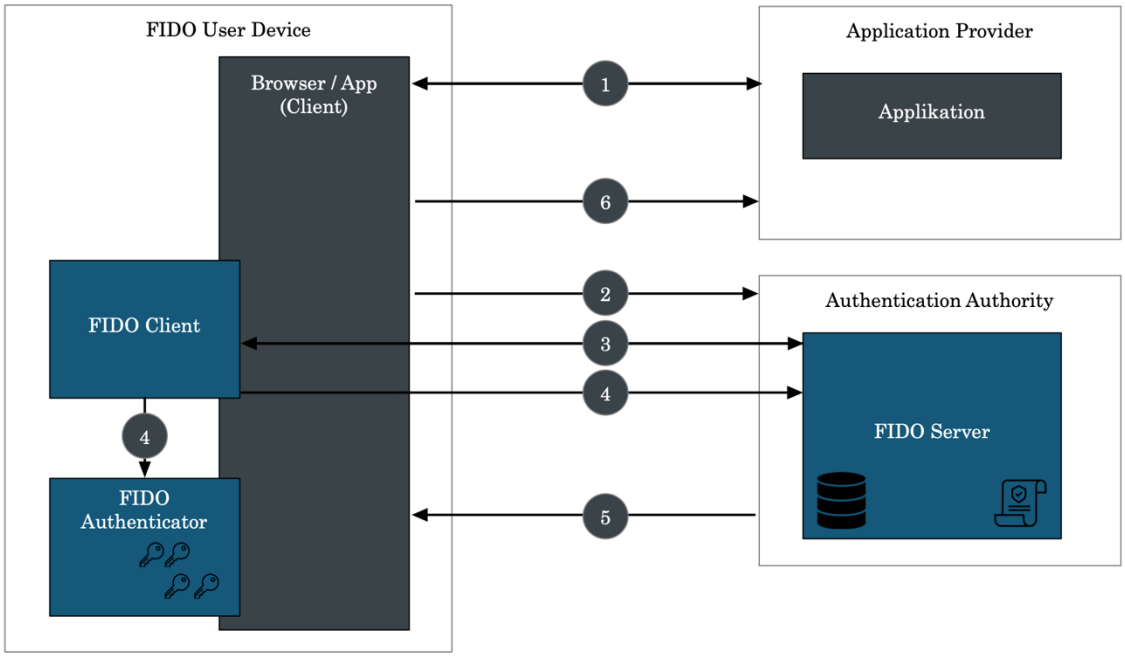

Abbildung 23: FIDO und Federation (in Anlehnung [FI17])

Die Schritte der Authentifizierung gliedern sich wie folgt:

1. Der Benutzer/die Benutzerin ruft über den Browser / App (Client) den Application Provider auf, der die Applikation darstellt. Dieser leitet den Client, mit einer Anfrage zur Authentifizierung, zur Authentication Authority um.
2. Der Client startet den Authentifizierungsvorgang an der Authentication Authority.
3. Die Authentication Authority startet über den Client eine FIDO basierte Authentifizierung.
4. Der FIDO Client ruft den FIDO Authenticator auf und sendet der Authentication Authority die FIDO Response zurück.
5. Die Authentication Authority validiert die Authentifizierung und den Benutzer/die Benutzerin. Bei erfolgreicher Prüfung wird eine Bestätigung nach dem Authentifizierungsstandard (SAML oder OpenID Connect) ausgestellt und der Client zum Application Provider gesendet.

2.2 Sicherheitsmechanismen der mobilen Plattformen Apple iOS bzw. Google Android

Mit der Einführung des Apple iPhone 5S und dem A7 Chip ging ebenfalls die Einführung eines Co-Prozessors einher, der für die Ablage von besonders sensiblen Informationen verwendet wird [MSW16]. Dieses Secure Element oder auch Secure Enclave ist ein vor Manipulation geschützter Bereich am System On a Chip (SOC) des mobilen Endgeräts. Damit lässt sich ein unbefugtes Lesen, Ändern oder Extrahieren von sensiblen Daten unterbinden. Zu den sensiblen Daten gehören unter anderem die biometrischen Merkmale für die Entsperrung der Geräte als auch kryptographische Schlüssel, die zur Authentifizierung oder Verschlüsselung verwendet werden. Hersteller von mobilen Endgeräten gehen in den letzten Versionen noch einen Schritt weiter und entkoppeln nicht nur den Speicherbereich, sondern setzen einen dedizierten Chip für sensible Operationen ein.

Unter iOS sieht der Secure Enclave Processor (SEP), wie der Zertifizierungsbericht [Ap19] des NIST (National Institute of Standards and Technology) zeigt, wie in Abbildung 24 dargestellt aus. Mobile Apps können über die APIs die Secure Enclave ansprechen und kryptographische Operationen, wie eine Schlüsselgenerierung, auslösen bzw. steuern. So lässt sich angeben, ob eine Berechtigungserteilung durch den Benutzer/die Benutzerin erforderlich ist und welche Anwendungen auf die Objekte zugreifen dürfen.

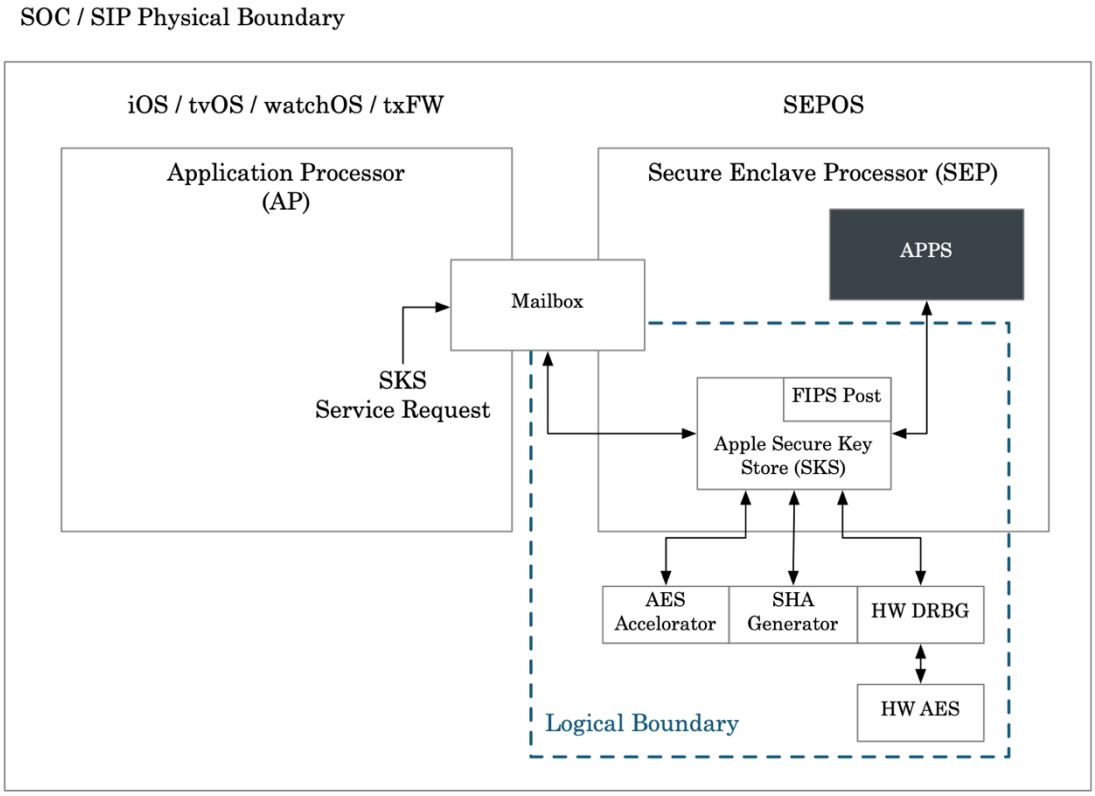

Abbildung 24: Secure Enclave Processor (SEP) (in Anlehnung an [Ap19])

Mit dem Android KeyStore System [An20b] hat Google für das Betriebssystem Android eine vergleichbare Architektur geschaffen. Kryptographisches Schlüsselmaterial wird hierbei ebenfalls vor Veränderung oder Extraktion in einem Trusted Execution Environment (TEE) bzw. Secure Element (SE) geschützt. Mit Android 9 (Pie) oder aktueller wurde mit StrongBox [An20b] der nächste Schritt gesetzt. Dabei wurde versucht, restriktivere Vorgaben an die verschiedenen Hersteller von mobilen Endgeräten mit Android zu geben. Diese haben in der Vergangenheit unterschiedliche Ansätze für die Implementierung des SOC, im speziellen von dem TEE bzw. SE verfolgt und keine starke Trennung zwischen der sicheren Umgebung und dem eigentlichen Mikroprozessor geschaffen. Mit der StrongBox Erweiterung gingen Anforderungen an eine eigene CPU, einem sicheren Speicher und der Einführung von Verfahrung zur Unterbindung von Sideloading Angriffen einher. Google selbst hat mit der Veröffentlichung von Android 9 und den

Google Pixel Geräten [Ti18] eine Version eines Hardware Security Modules für seine Geräte vorgestellt. Dieser Titan M Chip soll, vergleichbar mit dem Secure Enclave Processor unter iOS, Operationen für die Speicherung und Verwendung von kryptographischen Objekten, entkoppelt vom SOC durchführen.

2.3 Hardware Security Modul

„Die Funktion eines HSM kann man sich zunächst wie die einer Smartcard vorstellen. Auf dem HSM ist ein geheimer Schlüssel unauslesbar gespeichert." [Sc16]

Ein Hardware Security Module (HSM) ist ein manipulationssicherer Server bzw. eine Steckkarte für einen Server, über den kryptografische Operationen durchgeführt werden können. [TJ14]

Zu diesen gehören unter anderem:

- Generierung und Speicherung von symmetrischen Schlüsseln oder asymmetrischen Schlüsselpaaren,
- Berechnung von Hash Algorithmen,
- Ver- und Entschlüsseln von Daten oder anderen Schlüsseln bzw. Schlüsselpaaren,
- Erstellen von digitalen Signaturen und
- Erzeugen von Zufallszahlen.

Der Einsatz eines HSM im Vergleich zur Verwendung von Software Security Modulen beruht im Wesentlichen darauf, dass die Objekte, die darin gespeichert werden, unter keinen Umständen entwendet werden können. Der Aufbau eines Hardware Security Moduls ist vergleichbar mit dem in Abbildung 24 dargestellten Aufbau des Secure Enclave Processor, wenn auch um ein Wesentliches umfangreicher. So besitzen sie einen höheren Manipulationsschutz, damit bei einem böswilligen Eindringen durch Dritte eine Löschung des Systems durchgeführt wird. Dafür bieten die Systeme eine definierte Schnittstelle mit den notwendigen

Operationen an. Eine verbreitete Methode, neben spezifischen Schnittstellen der Hersteller, ist der Zugriff über PKCS#11. [PK15]

2.4 Fazit

Most of the applications developed in the last few years are supporting OpenID Connect. Ninety-two percent of the 8 billion+ authentication requests Microsoft Azure AD handled in May 2018 were from OpenID Connect–enabled applications. [Sr14]

Um ein komfortables Single Sign ON (SSO) für den Benutzer zu erreichen, steht der Designer der Anwendung vor der Entscheidung zwischen der Nutzung von SAML oder OpenID Connect. SAML auf der einen Seite ist ein etabliertes Protokoll mit einer breiten Unterstützung. Auf der anderen Seite steht das Protokoll OpenID Connect, welches aktuell durch einige Erweiterungen, wie dem RFC zum Token Binding (siehe Abschnitt 2.1.2.5 bzw. 2.1.2.6) oder auch Sicherheits-Empfehlungen [SR20b] für den Einsatz in sensiblen Umgebungen wie dem Finanz Sektor [DP18] bzw. aufwartet.

Nachfolgend ist der Ablauf des SAML *SSO Web Profile* mit *HTTP Post Binding* dargestellt. Die Kommunikation der Anmeldung läuft stets über den Client ab, welcher initial den Service Provider aufruft und einen Redirect (SAML *Authn Response*) für die Anmeldung am Identity Provider erhält. Der Austausch der Daten zwischen dem Service Provider und Identity Provider läuft hierbei vollständig über den Browser ab und damit indirekt („Front Chanel“). Dies erfolgt über HTML Formulare, die automatisch abgesendet werden. Alternativ kann das *HTTP Redirect Binding* verwendet werden. Dabei werden die Nachrichten über die HTTP URL Query String transferiert, sofern die Daten der SAML Assertions nicht zu groß sind.

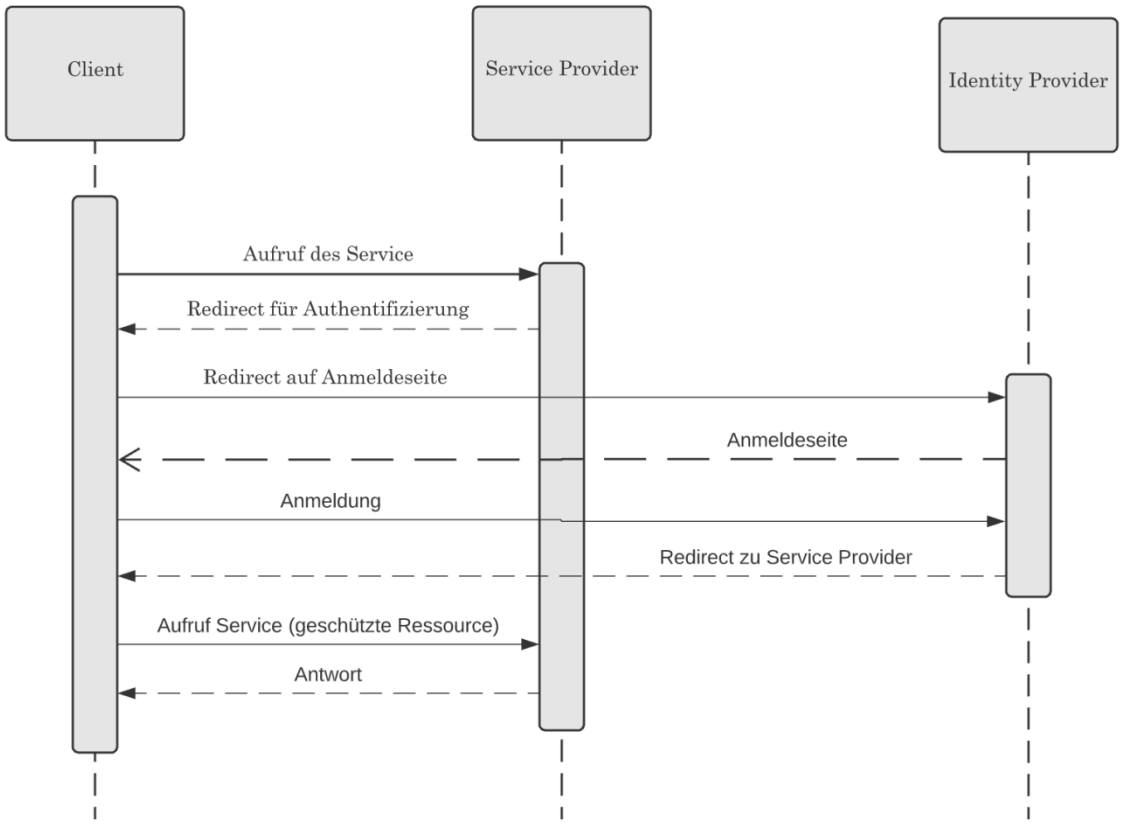

Abbildung 25: SAML Ablauf Anmeldung

Zum Vergleich dazu ist in der nachfolgenden Abbildung 26 der Ablauf von OpenID Connect nach der Empfehlung des RFC 8252 für native Apps (siehe Abschnitt 2.1.2.3) dargestellt. Wie der Name schon andeutet, ist der Ablauf speziell für den Einsatz in mobilen Anwendungen konzipiert. Der Client startet die Anmeldung über den Browser des Betriebssystems und nachdem die Anmeldung am OpenID Connect Provider durchgeführt wurde, wird per Intent die App wieder aktiviert und der Authorization Code gegen den Access Token getauscht.

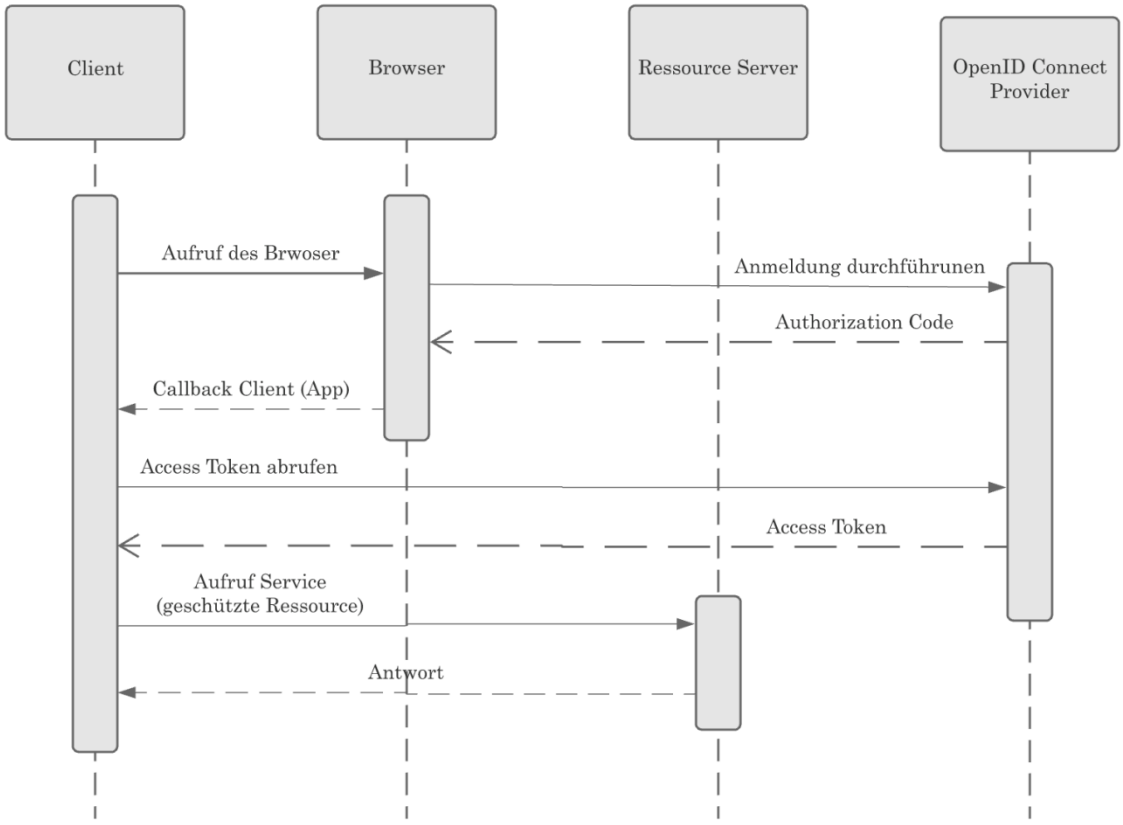

Abbildung 26: OpenID Connect Authentifizierung

Im Vergleich der beiden Protokolle SAML 2.0 und OpenID Connect sieht man hinsichtlich des Entstehungszeitpunktes (2005) von SAML 2.0, dass die heutigen technischen Möglichkeiten der mobilen Endgeräte nicht berücksichtigt werden. Die Abläufe der Anmeldung gemäß des SAML Protokolls lassen sich zwar über Workarounds lösen, können allerdings, wie nachfolgend anhand eines Beispiels erörtert wird, zu Nachteilen führen.

Ein Workaround kann z.B. sein, dass die Anmeldung direkt in der App implementiert wird (nativ) oder in einer Web View, falls es sich um eine hybride Anwendung handelt. Damit löst man das Problem, dass die Anmeldung über SAML durchgeführt werden kann, widerspricht aber der Empfehlung des RFC 8252 (siehe Abschnitt 2.1.2.3), dass die Anmeldung über den System Browser erfolgen soll, um nicht an vertrauenswürde Credentials des Benutzers/der Benutzerin kommen zu

können. Darüber hinaus kann damit kein komfortables Single Sign On (SSO) erreicht werden, da man sich stets innerhalb der mobilen Anwendung befindet und die Anmeldung nicht über mehrere geteilt werden kann, wie es im Browser und bei OpenID Connect der Fall ist.

Neben der Berücksichtigung von mobilen Anwendungen bereits bei der Konzeption von OpenID Connect bzw. OAuth 2.0 bietet es noch zwei weitere wesentliche Vorteile, einerseits kann zu einem späteren Zeitpunkt die österreichische Lösung der E-ID (siehe Abschnitt 2.1.3.2), als auch mit FIDO (siehe Abschnitt 2.1.4) integriert werden. Zum Zeitpunkt der Erstellung dieser Arbeit war die österreichische Lösung der E-ID in Umsetzung und kann daher nicht in der experimentellen Umsetzung berücksichtigt bzw. integriert werden. Sofern der OpenID Connect Provider und die Browser der mobilen Endgeräte FIDO und das WebAuthn Protokoll unterstützen, kann man darüber hinaus eine starke Authentifizierung [SM18b] bei der Anmeldung nutzen.

3. Experimentelle Umsetzung

In diesem Kapitel wird die konzeptionelle bzw. experimentelle Umsetzung eines Frameworks zur Ver- und Entschlüsselung von Daten erarbeitet. Ausgehend vom theoretisch aufbereiteten Wissen im vorangegangenen Kapitel 2 wird zunächst das Framework in Abschnitt 3.1 konzipiert. Dabei werden die Anwendungsfälle und deren Abläufe definiert. Eckpunkte dieses Frameworks sind (1) Authentifizierung basierend auf einem Standard zur Authentifizierung, (2) Berücksichtigung der Sicherheitsmechanismen der mobilen Endgeräte zur Verwaltung der kryptographischen Objekte und (3) Verwendung eines Hardware Security Modules serverseitig zur sicheren Speicherung der Schlüsselobjekte für den Benutzer/die Benutzerin. In Abschnitt 3.2 wird auf Basis des dargestellten Frameworks ein Prototyp experimentell umgesetzt und das Framework auf Machbarkeit überprüft. Gemäß der Auswahl des Protokolls (siehe Abschnitt 2.4) wird ein OpenID Connect Provider für die Authentifizierung und Autorisierung eingesetzt. Zusätzlich werden Sicherheitsmechanismen der mobilen Betriebssysteme in Betracht gezogen und in die Umsetzung der Android App integriert.

3.1 Remote Crypto Framework

Der Benutzer/die Benutzerin soll von einer zentralen und vertrauenswürdigen Instanz einen symmetrischen Schlüssel bzw. ein Schlüsselpaar beziehen können, um damit beliebige Daten ver- und entschlüsseln zu können.

Folgende Kernelemente der Analyse aus Abschnitt 2.4 werden dabei vom Framework berücksichtigt:

1. Die Authentifizierung wird gemäß dem OpenID Connect Standard durchgeführt, unter Berücksichtigung von Sicherheitsmechanismen, die das Entwenden der Bearer Tokens verhindert.
2. Die Schlüssel werden lokal am Smartphone abgelegt und die Daten können damit vor dem Verlassen des Smartphones verschlüsselt werden. Die lokalen

Kopien der Schlüssel werden auf den Endgeräten im Secure Element (SE) abgelegt und können nur durch Authentifizierung des Benutzers/der Benutzerin verwendet werden.

Darüber hinaus wird im Konzept des Frameworks berücksichtigt, dass die zentrale Schlüsselablage der kryptographischen Objekte der Nutzer und Nutzerinnen in einem Hardware Security Module (HSM), zum Schutz vor einer Kompromittierung, erfolgt.

Aus diesen Punkten wird die in Abschnitt 1.2 dargestellte Zielsetzung um die nachfolgenden Elemente (siehe Abbildung 27) erweitert. Der Client verwendet einen OpenID Connect Provider für die Anmeldung (Login) zur Security Anwendung und kann damit seine privaten bzw. sensiblen kryptographischen Objekte (Schlüssel), welche in einem Hardware Security Module (HSM) abgelegt sind, abrufen.

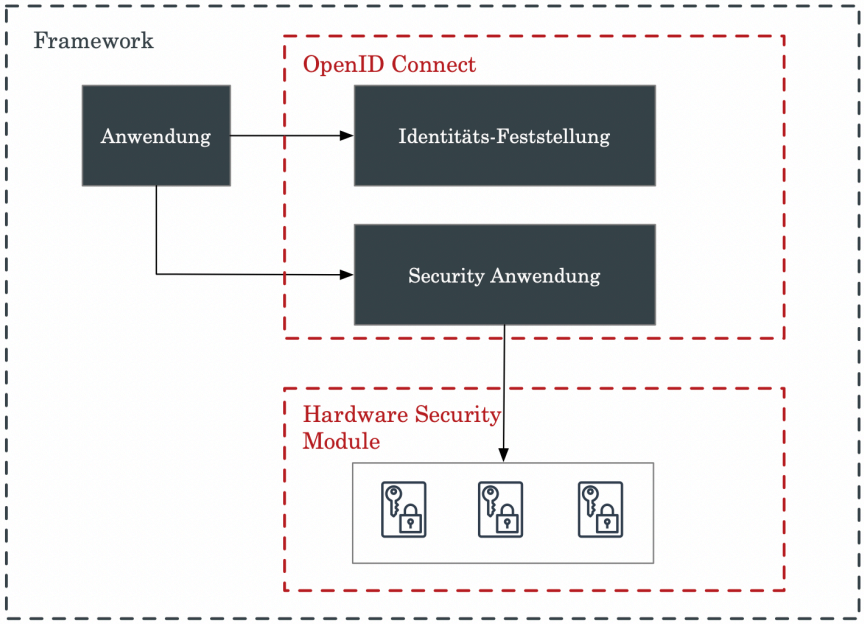

Abbildung 27: Entwurf des Frameworks

3.1.1 Überblick der Anwendungsfälle und Zustände

Das Remote Crypto Service gliedert sich in die in Abbildung 28 aufgelisteten Use-Cases (1) Login, (2) Registrierung, (3) Schlüssel abrufen und (4) Verwendung.

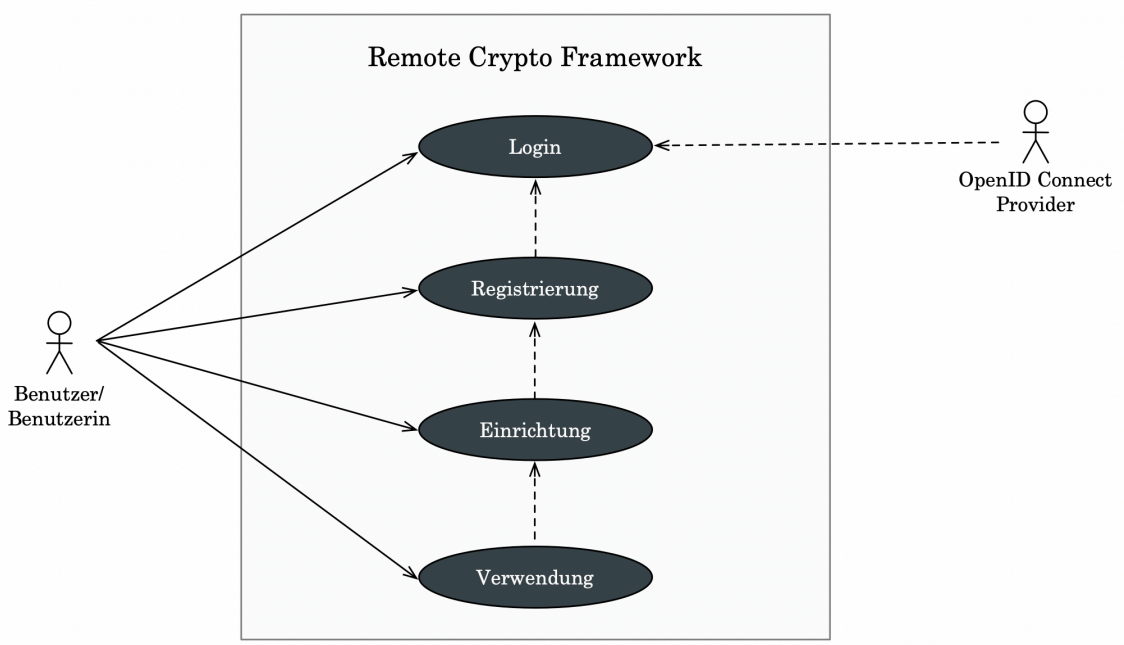

Abbildung 28: Use-Cases Remote Crypto Framework

Im ersten Use-Case (1) führt der Benutzer/die Benutzerin ein Login (Anmeldung) über eine OpenID Connect Provider durch. Die Anmeldung zur erstmaligen Identifikation des Benutzers/der Benutzerin erfolgt hierbei über das Protokoll OpenID Connect zur Authentifizierung. Die Anmelde- bzw. Benutzerdaten werden nach dem Login für die Registrierung (2) genutzt. In diesem Schritt wird die installierte App auf dem Gerät für den angemeldeten Benutzer/die angemeldete Benutzerin registriert. Dafür wird am lokalen Gerät ein asymmetrisches Schlüsselpaar generiert und bei der Registrierung bekannt gegeben. Nach der Registrierung wird für den Benutzer/die Benutzerin ein Schlüssel bzw. Schlüsselpaar erstellt, welcher im Use-Case (3) an den Benutzer/die Benutzerin bzw. die App retourniert wird. Nach erfolgreicher Speicherung des Schlüssels in der Secure Enclave kann dieser für den Verwendungszweck genutzt werden (4). Die

Verwendung der kryptographischen Schlüssel ist kein Bestandteil der Szenarien in denen dieses Framework zum Einsatz kommen kann.

Aus den Anwendungsfällen des Remote Crypto Frameworks lassen sich die in Abbildung 29 skizzierten Zustände (1) Status unbekannt, (2) Nicht angemeldet, (3) Angemeldet, (4) Registriert und eingerichtet und (5) Initialisiert einteilen. Die Zustände sind dabei aus Sicht der mobilen Anwendung dargestellt.

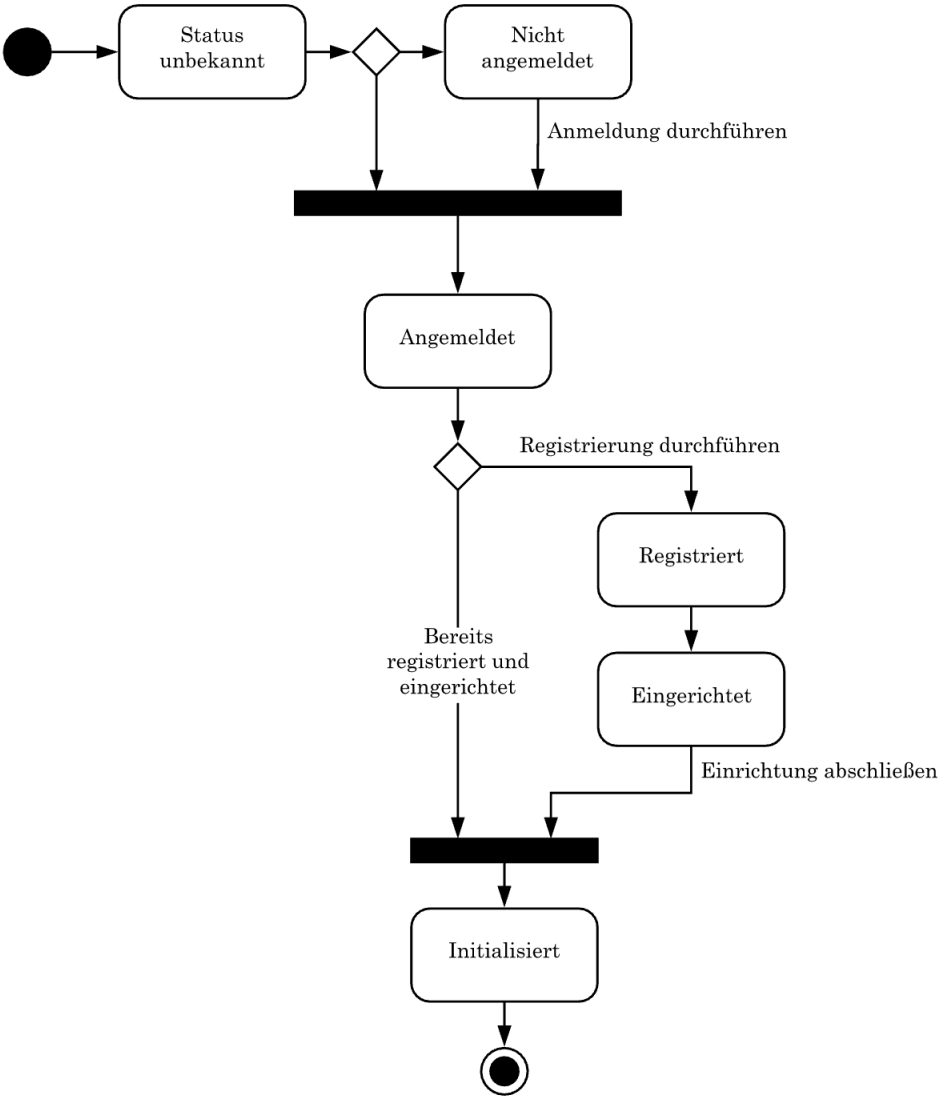

Abbildung 29: Zustände der App des Remote Crypto Framework

Die Abläufe für das Remote Crypto Framework starten beim Öffnen der Anwendung im Zustand (1) Status unbekannt – siehe nachfolgende Tabelle 1.

Status	**Beschreibung**	**Übergang**
Status unbekannt	Der Status der Anwendung ist zum Zeitpunkt des Öffnens unbekannt.	Nach Prüfung des Status wird entweder in den Status *angemeldet* oder *nicht angemeldet* gewechselt.
Nicht angemeldet	Der Benutzer/die Benutzerin hat sich noch nicht angemeldet oder die letzte Sitzung ist bereits abgelaufen.	Die Anmeldung über OpenID Connect bzw. den System Browser wurde erfolgreich durchgeführt und der Access/Refresh bzw. ID Token wurde abgeholt.
Angemeldet	Die Anwendung hat einen gültigen Access bzw. ID Token, mit dem sie Ressourcen aufrufen kann. Darüber hinaus wird geprüft, ob bereits eine Registrierung durchgeführt wurde	Ist eine Registrierung erforderlich, wird in den Zustand *Registriert* gewechselt, ansonsten ist der Benutzer/die Benutzerin bereits registriert und es wird in den Zustand *Initialisiert* gewechselt. Im Zuge der Registrierung wird ein lokales Schlüsselpaar am mobilen Endgerät generiert und mit dem Access/ID-Token der Benutzer/Benutzerin registriert.

Status	Beschreibung	Übergang
Registriert	Die Registrierung des Benutzers/der Benutzerin wurde abgeschlossen.	Mit dem bereits registrierten Schlüsselpaar wird der Schlüssel bzw. das Schlüsselpaar von der Security Anwendung verschlüsselt.
Eingerichtet	Der Schlüssel bzw. das Schlüsselpaar zur Ver- und Entschlüsselung wurde abgerufen und kann lokal abgelegt werden.	Der sensitive Schlüssel wird im Secure Element des mobilen Endgeräts abgelegt und die Einrichtung ist damit abgeschlossen.
Initialisiert	Die Anwendung ist eingerichtet und kann verwendet werden.	-

Tabelle 1: Zustände der App des Remote Crypto Framework

3.1.2 Abläufe des Remote Crypto Framework

Die *Anmeldung (Login)* und die *Registrierung* der mobilen Anwendung folgt dem in der Abbildung 30 dargestellten Ablauf. Eingangs wird der Status der Anwendung überprüft – hierbei wird überprüft, ob der Access-/ID-Token und somit das Login noch gültig ist. Wenn dies nicht mehr gegeben ist, wird der OpenID Connect Authentifizierungsprozess gestartet. Dazu wird der lokale Browser aufgerufen und der Login der OIDC Providers angezeigt. Nach erfolgreicher Anmeldung leitet der Browser, z.B. per Intent unter Android, wieder zurück zu der mobilen Anwendung und übergibt den Authorization Code. Mit diesem ruft die App den Access-/ID-Token vom OIDC Provider (Token Endpoint) ab.

Die OpenID Connect basierte Anmeldung hat dabei den Empfehlungen der Standards für native Anwendungen und dem für den Einsatz von PKCE, siehe Abschnitt 2.1.2.3 bzw. 2.1.2.4 zu folgen, um einerseits ein Single Sign On über

mehrere Apps hinweg zu gewährleisten, andererseits aber auch ein Mindestmaß an Schutz vor Dritt-Apps zu gewährleisten.

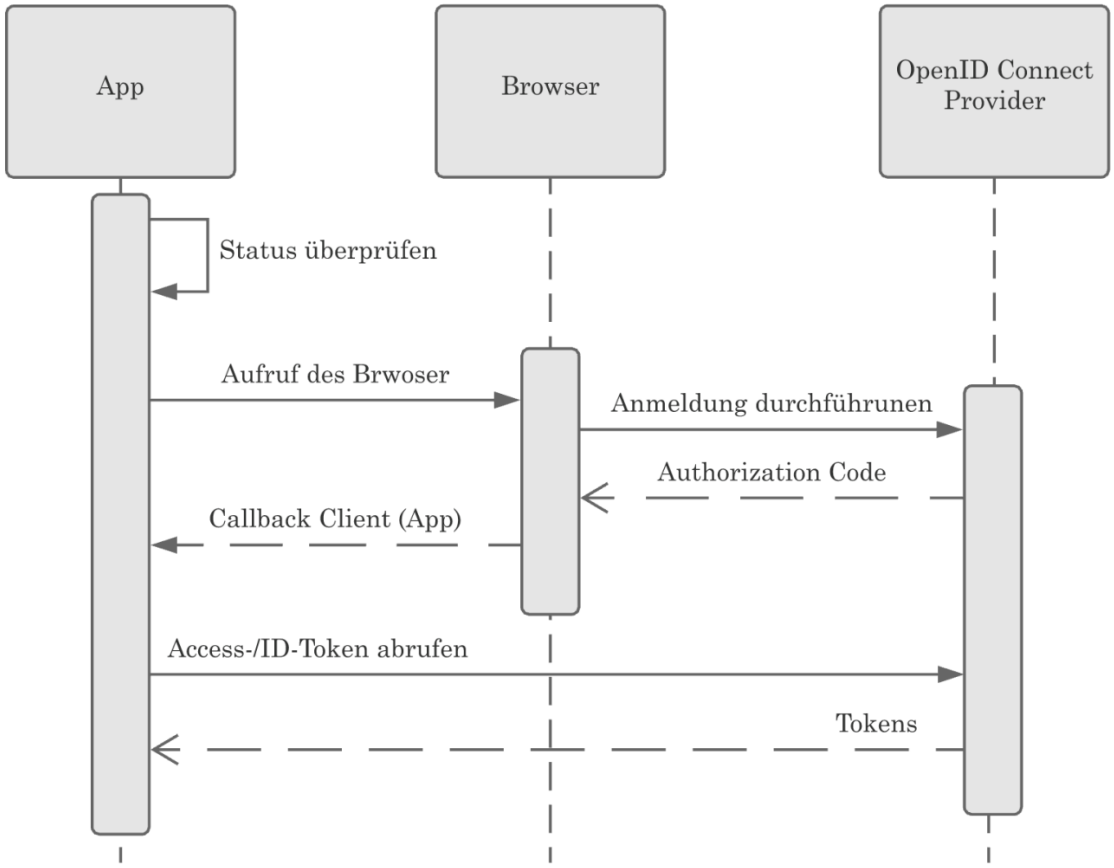

Abbildung 30: Anmeldung der App des Remote Crypto Frameworks

Im Anschluss zur Anmeldung erfolgt die Prüfung, ob:

1. ein lokales Schlüsselpaar (*TS*[1]) für den verschlüsselten Austausch mit der Security-Anwendung vorhanden ist und

[1] TS; Transport Schlüsselpaar

1. der Schlüssel/das Schlüsselpaar für die Ver- und Entschlüsselung (*DS*[2]) vorhanden ist bzw.
2. der Benutzer/die Benutzerin die biometrische Authentifizierung aktiviert hat.

Bei erfolgreicher Prüfung ist die App *initialisiert* und kann für den gegebenen Einsatz verwendet werden. Ist eine *Registrierung und Einrichtung* erforderlich, wird ein lokales Schlüsselpaar (*TS*) benötigt, welches in der Secure Enclave generiert wird. Der öffentliche Schlüssel (*TS.pub*) von diesem Schlüsselpaar wird exportiert und gemeinsam mit Informationen über das aktuelle mobile Endgerät registriert. Serverseitig wird der Access-/ID-Token validiert und bei erfolgreicher Prüfung wird der öffentliche Schlüssel (*TS.pub*) für den Benutzer/die Benutzerin registriert.

Nach der *Registrierung* – siehe Abbildung 31 – wird der Schlüssel bzw. das Schlüsselpaar (*DS*) für den Benutzer/der Benutzerin in seiner/ihrer App *eingerichtet*. Dafür wird der zuvor registrierte öffentliche Schlüssel des Benutzers/der Benutzerin (*TS.pub*) in das Hardware Security Modul (HSM) importiert. Weiters wird bei einem neuen Benutzer/einer neuen Benutzerin ein neuer Schlüssel bzw. Schlüsselpaar (*DS*) im HSM generiert. Dabei ist zu beachten, dass für jeden Benutzer/jede Benutzerin nur ein dedizierter Schlüssel bzw. Schlüsselpaar (*DS*) existiert. Führt der Benutzer/die Benutzerin auf unterschiedlichen Geräten oder bei einer Neu-Installation der Anwendung auf dem mobilen Gerät durch und er/sie besitzt bereits einen *Daten-Schlüssel bzw. Schlüsselpaar DS*, wird das bestehende verwendet. Im Anschluss des Imports des TS.pub und optionaler Generierung des DS wird mit dem öffentlichen Schlüssel, der gerade durchgeführten Registrierung der DS aus dem HSM, verschlüsselt, exportiert und an die App übermittelt. In der App wird

[2] DS; Daten-Schlüssel bzw. Schlüsselpaar

daraufhin eine Entschlüsselung des DS in der Secure Enclave über den privaten Transport Schlüssel (TS.priv) durchgeführt.

Zusammengefasst können für einen Benutzer/eine Benutzerin damit folgende Objekte existieren[3]:

1. Ein (1) Objekt des Daten-Schlüssels bzw. Schlüsselpaar (DS) und
2. n Transport Schlüsselpaare TS für einen Benutzer/eine Benutzerin.

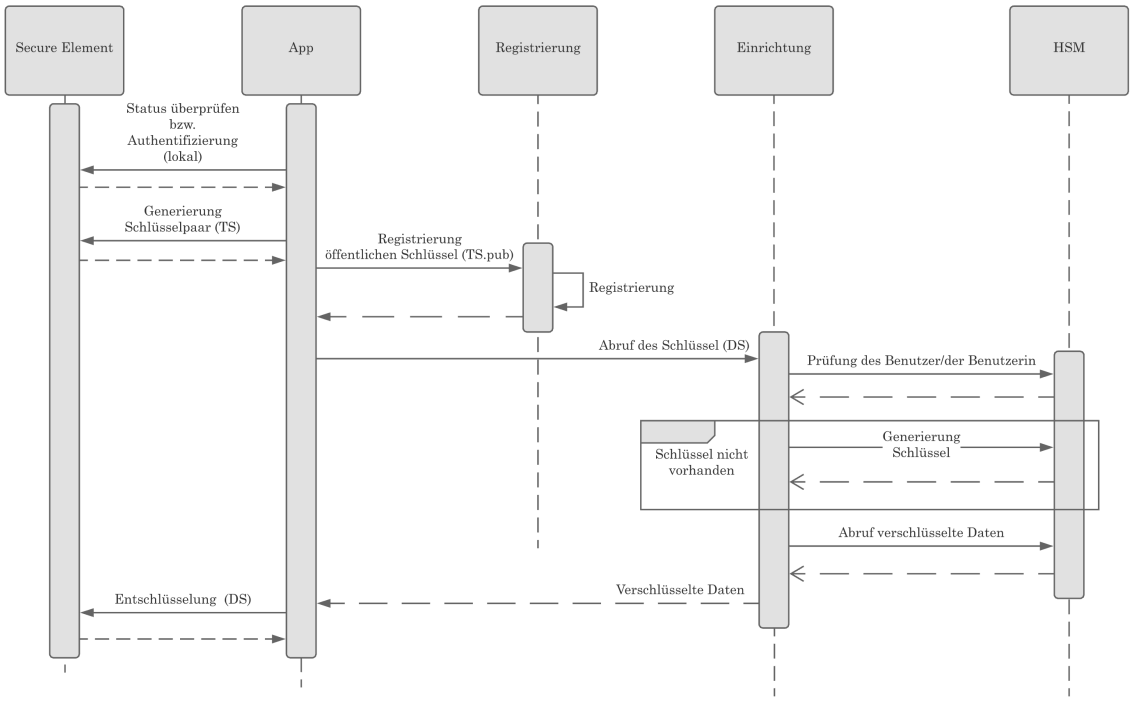

Abbildung 31: Registrierung und Einrichtung der App des Remote Crypto Frameworks

Nach Abruf des symmetrischen Schlüssels kann dieser für eine Ver- und Entschlüsselung von Daten verwendet werden. Damit diese Funktionalität getestet

[3] Schlüsselwechsel des Daten-Schlüssels bzw. Schlüsselpaar (DS) ist aktuell nicht berücksichtigt

werden kann, kommt die letzte Komponente das Daten-Service zum Zug. Dieses kann beliebige verschlüsselte Daten entgegennehmen und liefert alle für den Benutzer/die Benutzerin bereits hochgeladenen Daten zurück. Die mobile App kann damit alle Daten von einem Benutzer/eine Benutzerin abrufen und darstellen.

3.2 Machbarkeitsprüfung

In diesem Abschnitt wird das konzipierte *Remote Crypto Framework* auf deren Machbarkeit überprüft. Dazu werden sowohl die serverseitigen Komponenten als auch die Client Anwendung für einen ausgewählten Anwendungsfall experimentell umgesetzt.

Das Framework beschreibt abstrakt, wie Abläufe aussehen und dieses strukturiert sind. In der Überprüfung der Machbarkeit wird nachfolgender Anwendungsfall herangezogen:

Der Benutzer möchte von einer zentralen Stelle einen symmetrischen Schlüssel zur Ver- und Entschlüsselung von Daten am Smartphone beziehen.

Somit werden die folgenden Parameter des Frameworks für die Anwendung konkretisiert:

1. Algorithmen
    A. Daten Schlüssel (DS) - AES (Advanced Security Framework) und
    B. Transport Schlüsselpaar (TS) - RSA (Rivest Shamir Adelman).
2. Verwendung zur Ver- und Entschlüsselung von Daten und
3. Entwicklungsumgebung für die mobile Anwendung Google Android.

Dieser Prototyp wird in weiterer Folge als *Remote Crypto Service (RCS)* bezeichnet, um dies vom Framework abzugrenzen. Teile des Quellcodes sind auszugsweise im Anhang A und B aufgelistet.

3.2.1 Komponenten und Schnittstellen des Remote Crypto Service

Das in Abschnitt 3.1 dargestellte Remote Crypto Framework wird anhand der in Abschnitt 3.1.2 dargestellten Abläufe in die Komponenten unterteilt:

1. Secure-HSM-App,
2. OpenID Connect Provider,
3. Secure-HSM-Registration,
4. Secure-HSM-Key und
5. Data-Service.

In der Abbildung 32 sind diese Komponenten schematisch dargestellt, dabei repräsentiert jede Komponente einen Anwendungsfall, welcher in Abschnitt 3.1 aufgelisteten wurde:

1. Anmeldung,
2. Registrierung,
3. Einrichtung und
4. Verwendung.

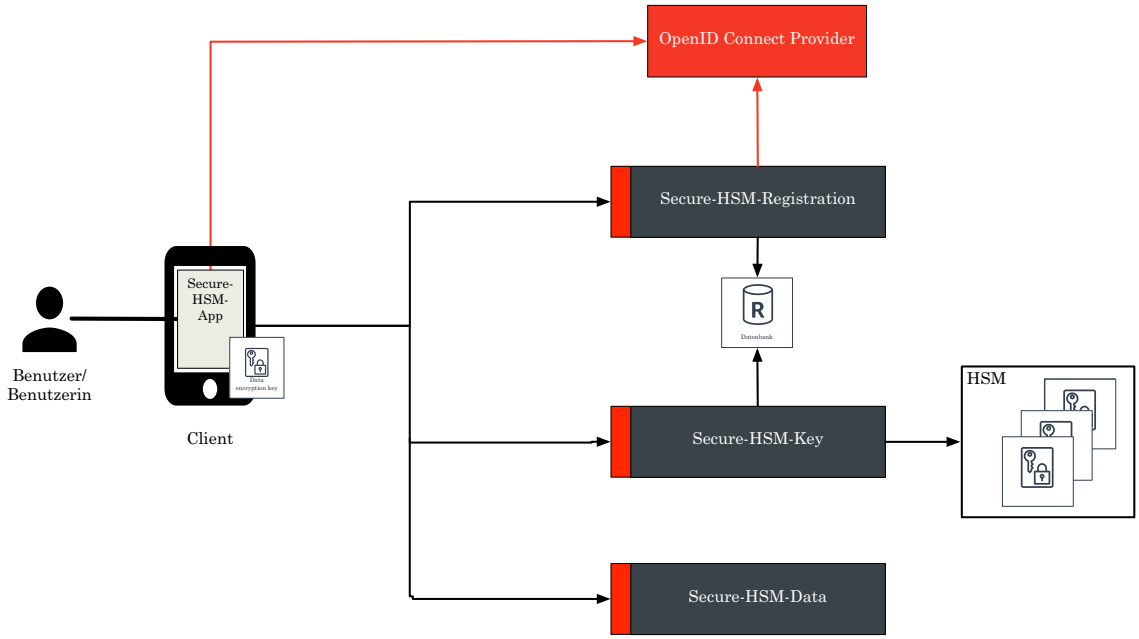

Abbildung 32: Komponenten des Remote Crypto Service

Komponente	Anwendungs-fall	Beschreibung
Secure-HSM App	-	Die mobile App unter Google Android implementiert ein rudimentäres User-Interface für die Anmeldung, Registrierung und die Überprüfung der Nutzung des symmetrischen Schlüssels (DS) zur Ver- und Entschlüsselung von Daten.
OpenID Connect Provider	*Anmeldung*	Die Anmeldung erfolgt über den Cloud Service Provider OKTA.
Secure-HSM-Registration	*Registrierung*	Diese Komponente führt die Registrierung des Benutzers/der Benutzerin durch: 1. Prüfung ob der Benutzer/die Benutzerin bereits registriert ist 2. Anlegen der neuen Registrierung und optional des Benutzers/der Benutzerin
Secure-HSM-Key	*Einrichtung*	Die Secure-HSM-Key Komponente steuert die kryptografischen Operationen im Zusammenspiel mit dem Hardware Security Modul. Die Secure-HSM-App kann über die Schnittstelle der Komponente den Daten-Schlüssel (*DS*) in verschlüsselter Form abrufen. Dafür werden die registrierten öffentlichen Schlüssel (*TS.pub*) im HSM gespeichert und der symmetrische *DS* in verschlüsselter Form (gewrapped) exportiert.
Secure-HSM-Data	*Verwendung*	Von dieser Komponente werden verschlüsselte Daten empfangen bzw. an den Client (Secure-

Komponente	Anwendungs-fall	Beschreibung
		HSM-App) für einen Benutzer/eine Benutzerin ausgeliefert.
HSM	-	Im HSM werden die öffentlichen RSA Schlüssel (*TS.pub*) der registrierten Benutzer/Benutzerinnen und die zugehörigen AES (*DS*) Schlüssel abgelegt.

Tabelle 2: Komponenten des Remote Crypto Service

Die Prüfung der Authentifizierung des Benutzers/der Benutzerin erfolgt über alle der in Abbildung 32 dargestellten bzw. in Tabelle 2 aufgelisteten Komponenten über die Autorisierung des Access-/ID-Tokens der Anmeldung.

3.2.2 Schnittstellen des Remote Crypto Frameworks

Die Registrierung und das Schlüssel-Service basieren auf APIs gemäß dem REST Standard [NF11]. Für die Registrierung werden die Identifikationselemente des ID-Token um nachfolgende Informationen erweitert:

1. Öffentlicher Schlüssel
2. Geräte-Informationen

Der öffentliche Schlüssel wird aus dem Secure Element des Gerätes exportiert und gemeinsam mit den Geräte-Informationen an das Service zur Registrierung übermittelt.

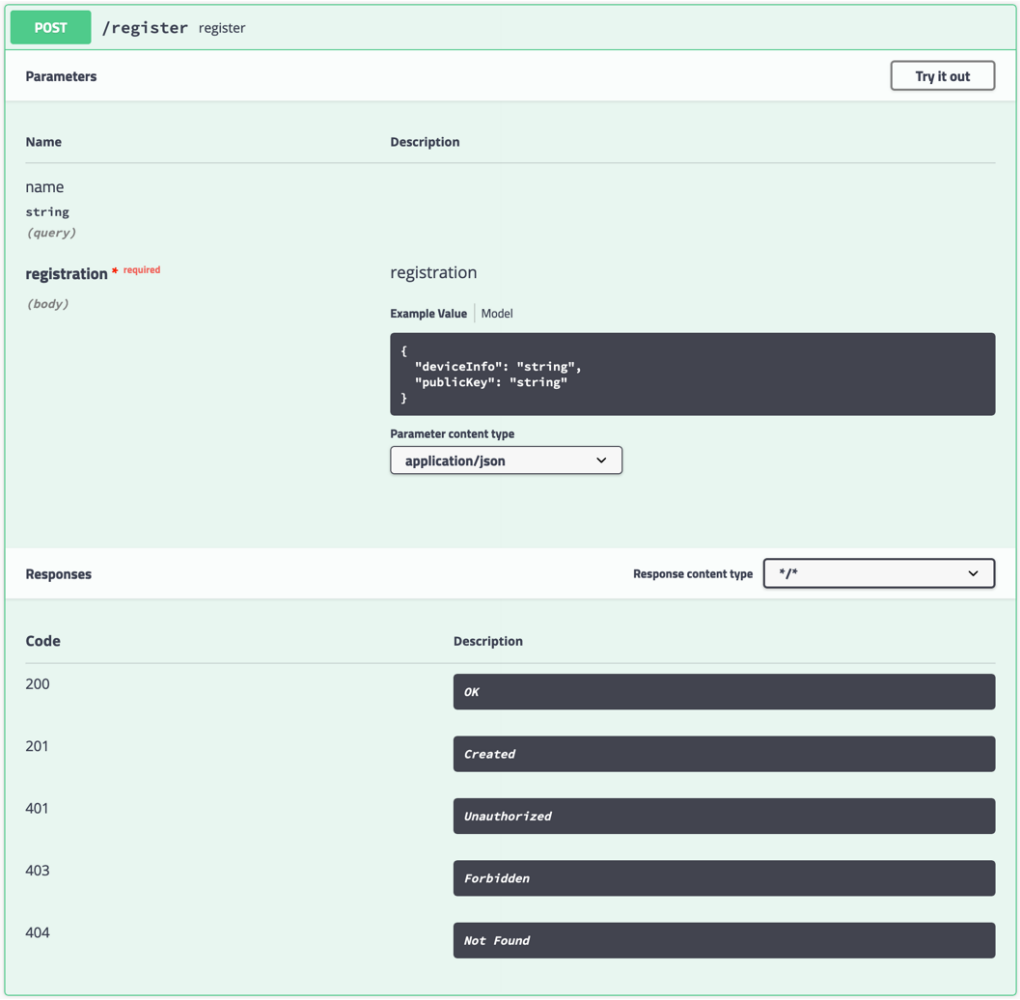

Abbildung 33: API Secure-HSM-Registration Service

Der nachfolgende Auszug (Listing 11) aus dem Service zeigt einen beispielhaften Aufruf des Service für eine Registrierung des öffentlichen Schlüssels.

```
1.  Received [POST /secure-hsm-registration/register HTTP/1.1
2.  Authorization: Bearer
3.  eyJraWQiOiJWQ1ZVNjZrRm5iM1U0VlVYOG05R3F6dklkV3JvN3ZyeXhqZjlOOFpmTTkwIiwiYWxnIjoiUl
4.  MyNTYifQ.eyJzdWIiOiIwMHV4enBsMjZrSUZWMUVFbzM1NiIsIm5hbWUiOiJNYXJpbyBHbGFzZXIiLCJ2Z
5.  XIiOjEsImlzcyI6Imh0dHBzOi8vZGV2LTg1NDU2Ni5va3RhLmNvbSIsImF1ZCI6IjBvYTIzb3VnbXVhRDZ
6.  Va1hnMzU3IiwiaWF0IjoxNTgyOTMxNDQ3LCJleHAiOjE1ODI5MzUwNDcsImp0aSI6IklELk5wZm1MUWVPZ
7.  ms0eS1kMVRZNGxvSkFuMG8zTEwxTHhJZ0VvZVBVLS04bHciLCJhbXIiOlsicHdkIl0sImlkcCI6IjAwb3h
8.  6b3Q2M3dFa3ZGWHRMMzU2Iiwibm9uY2UiOiIycEJNYTNEeFh3eWxqdUFLWnpxVjBRIiwicHJlZmVycmVkX
9.  3VzZXJuYW1lIjoibWFyaW8uZ2xhc2VyQG1haWwuZmVybmZoLmFjLmF0IiwiYXV0aF90aW1lIjoxNTgyOTM
```

```
10. wOTQ1LCJhdF9oYXNoIjoiVW9Ddi02YVo1bmtrcXA5eEQ3ZmVkUSJ9.SBftIA4NmGKwB17BCjFohj9sGYbT
11. 6LzwGDnz0URA59JQU2Os-
12. LPTlEvpggGNjphjZq8gtQ2R4X1uvnOt7rVF6OEDSoKuUmJ952sBo_06O0Q9Hc_xaIfcabDoHt1upGfDofY
13. 1RAYWtKBgoiUjTZ37DX44IvkLwzSIkdNTOEXLlW-9Pfdb-B6W_TStnDEE6n9KU_rHCeM-
14. jh_ooLjPhLtbwKWTP3NpcRpXeE3ymjWGHJyo1pERCgeOBxxpsmubM4cizrEx_1kgTe5zp3L-
15. 4cYuhB0IugDkbBKCqB8u6f_qWsKwjefECYy2p2jVzqgcjUhsl3RQp7HtCINMniYEOF2fNQ
16. Content-Type: application/json; charset=utf-8
17. Content-Length: 463
18. Host: 10.0.2.2:9200
19. Connection: Keep-Alive
20. Accept-Encoding: gzip
21. User-Agent: okhttp/4.2.2
22.
23. {
24. "publicKey":"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA2IdrzkyL0KG+6dbNF0mfd80G0
25. iZ1TM+nQNJEAcR0dBsv4ty1Nad4ON0vCnSkRFZhO/FPleWEjO56X4f5D9I+G4aiAnmYt3ZStRmYDa6E/cL
26. NM5W+JfG1fsn+kenmzFioGk9Zg2JVXR7DDvYUscQt7uw9h4l2WaF2awbDzJd0cNfBnptZJBcg3DLUBhT1d
27. 52HI8ai11dt98vhvcAQRLusFgpNZ4twxQg3ASgHteJ349N1V3xUssKEfC8Mkpn+3GAkCSvULfokUTg1xcC
28. 0rvYeFaO4yeozfgiqTeKxTDRpK1rtXq1eKVNcuHIjw4oMamDHlIV+MVC3japnHYjPqfRAUwIDAQAB",
29. "deviceInfo":"Google / Android SDK built for x86 / 10"
30. }
31. ]
```

Listing 11: Beispiel Registrierung (Remote Crypto Service)

Aus dem ID-Token werden die Informationen des Benutzers/der Benutzerin und der Anmeldung entnommen und geprüft. Dazu gehören die Benutzeridentität anhand der E-Mail-Adresse (*preferred_username)*, wie auch der Zeitpunkt der Anmeldung und der Zeitpunkt der nächsten Anmeldung.

Nach der erfolgreichen Registrierung kann das *Secure-Key-Service* zur Einrichtung der App aufgerufen werden, um den zugehörigen Ver- und Entschlüsselungsschlüssel abzurufen. Dafür wird an das Service der registrierte öffentliche Schlüssel übermittelt. Für diesen wird nach erfolgreicher Prüfung der Registrierung der entsprechende zugehörige verschlüsselte symmetrische Schlüssel inkl. verschlüsselten Testdaten zur Prüfung übermittelt. Die Identifikation des Benutzers/der Benutzerin erfolgt über den *preferred_username* aus dem ID-Token.

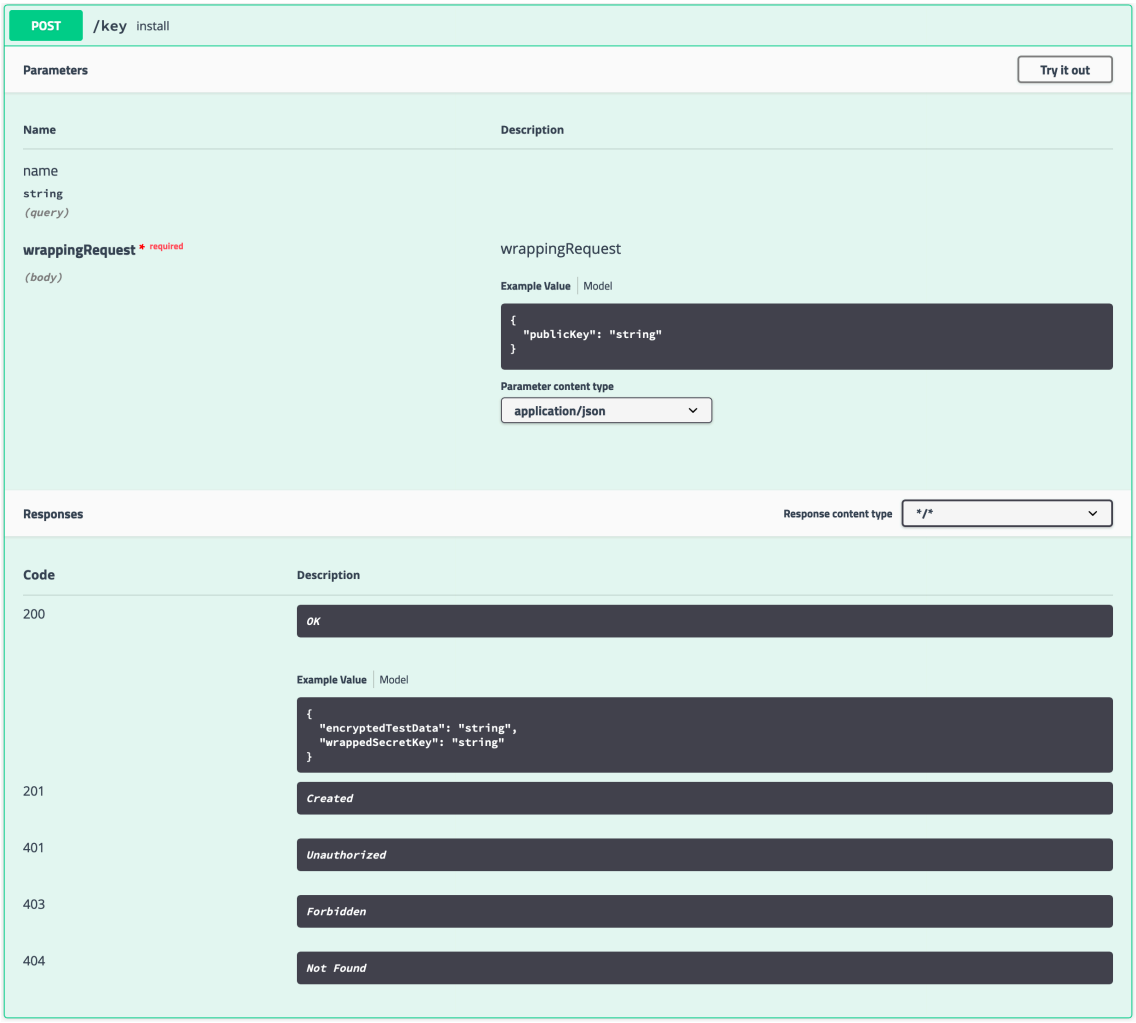

Abbildung 34: API Secure-HSM-Key Service

Als Ergebnis liefert das Service den verschlüsselten symmetrischen Schlüssel *(wrappedSecretKey)* und mit diesem verschlüsselte Testdaten *(encryptedTestData)*, damit diese am Client entschlüsselt und damit der Vorgang überprüft werden kann.

Das letzte der drei Anwendungen dient einzig zur Überprüfung des Vorganges der Ver- und Entschlüsselung. Dafür wird für einen Benutzer/eine Benutzerin eine Ablage für verschlüsselte Daten bereitgestellt. Dieser/diese kann damit Daten an das Service hochladen und alle Daten abrufen. In diesem Anwendungsfall wird der Benutzer/die Benutzerin wieder über den ID-Token und den *prefered_username*

identifiziert wird. Damit ist wie auch bei den vorhergehenden Anwendungen sichergestellt, dass eine Person unabhängig von deren Gerät stets dieselben Daten erhält.

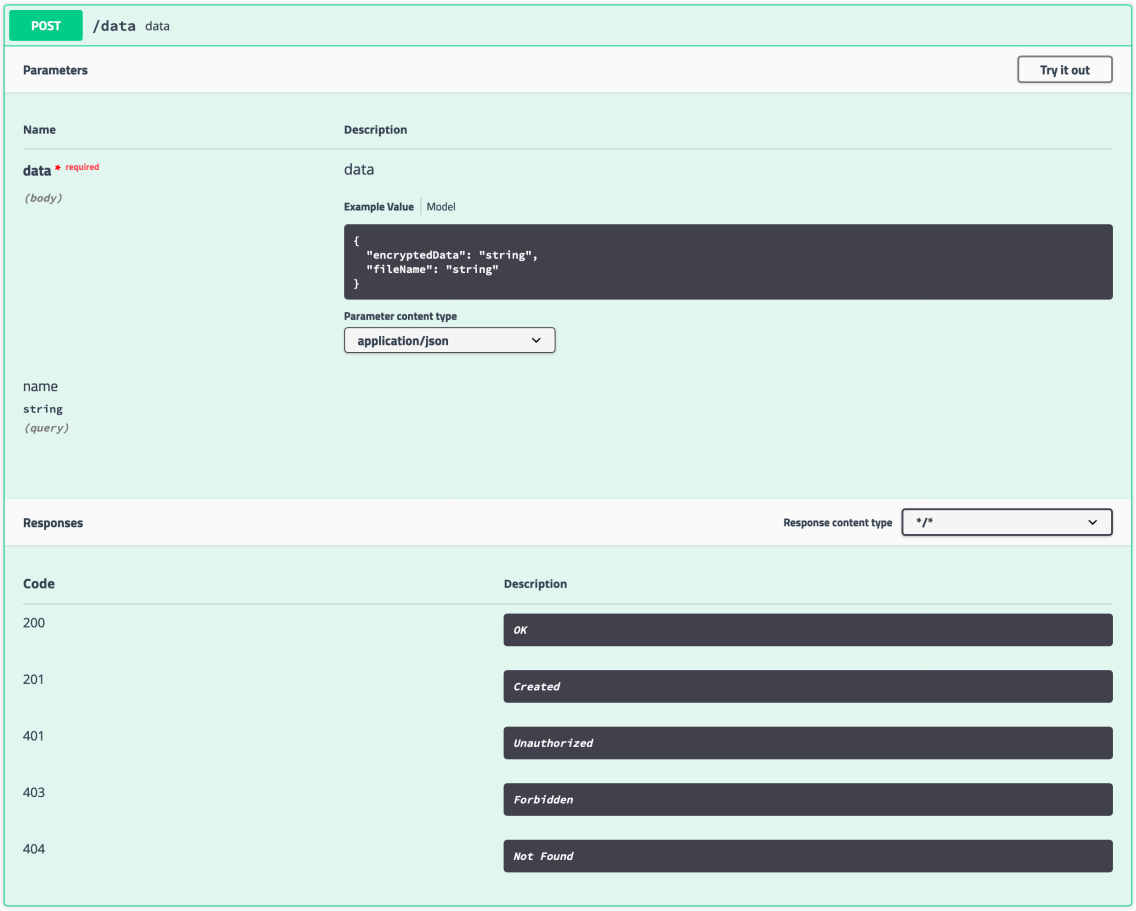

Abbildung 35: API Secure-HSM-Data Service

Die wesentlichen Auszüge aus dem Quellcode der serverseitigen Anwendungen des Remote Service sind im Anhang B aufgelistet.

3.2.3 Schnittstelle zum Hardware Security Module

Um ein herstellerunabhängiges Hardware-Security Module im Remote Crypto Service anzubinden, kann der Standard PKCS#11 [PK15] eingesetzt werden. Als

Abstraktion wird in der Umsetzung ein KeyStore [Ke20] verwendet. Über diese unabhängige Schnittstelle lassen sich die notwendigen Funktionen implementieren:

1. Speicherung der öffentlichen RSA Schlüssel (*TS.pub*)
2. Generierung der symmetrischen AES Schlüssel (*DS*)
3. Verschlüsseltes exportieren (wrapping) der *DS* mittels *TS.pub*

Die Speicherung der Schlüssel in Software kann durch Auswahl eines geeigneten Java Providers eines Hardware Security Moduls ausgetauscht werden.

3.2.4 Implementierung des Remote Crypto Service

Zur Prüfung der Machbarkeit des Remote Crypto Service wurde eine Android App (Secure-HSM-App) entwickelt, welche mit den Komponenten zur Anmeldung, Registrierung, Einrichtung und Verwendung kommuniziert. Für die Implementierung der Services wurde auf das Framework Spring Boot [Sp20b] gesetzt, welches eine einfache Möglichkeit zur Service-Entwicklung darstellt und die notwendigen Bibliotheken zur Absicherung der Anmeldung mittels OpenID Connect bereitstellt. Zur Simulation eines Hardware-Security Modules wurde ein Java KeyStore [Ke20] verwendet. Die Bereitstellung und der Test des gesamten Ablaufs erfolgte über eine lokale Entwicklungsumgebung bzw. ein Google Pixel 3 zur Überprüfung.

3.2.4.1 Umsetzung des OpenID Connect Provider

Als OpenID Connect Provider kommt das Cloud Service der OKTA Inc. [Ok20a] zum Einsatz. Dieser bietet eine einfache Einrichtung und Integration von mobilen Anwendungen an. Für die Anmeldung wurde hierfür eine mobile Anwendung eingerichtet (siehe nachfolgende Abbildung 36). Diese diente sowohl für die native App als auch die Backend Anwendungen des Secure-HSM Service.

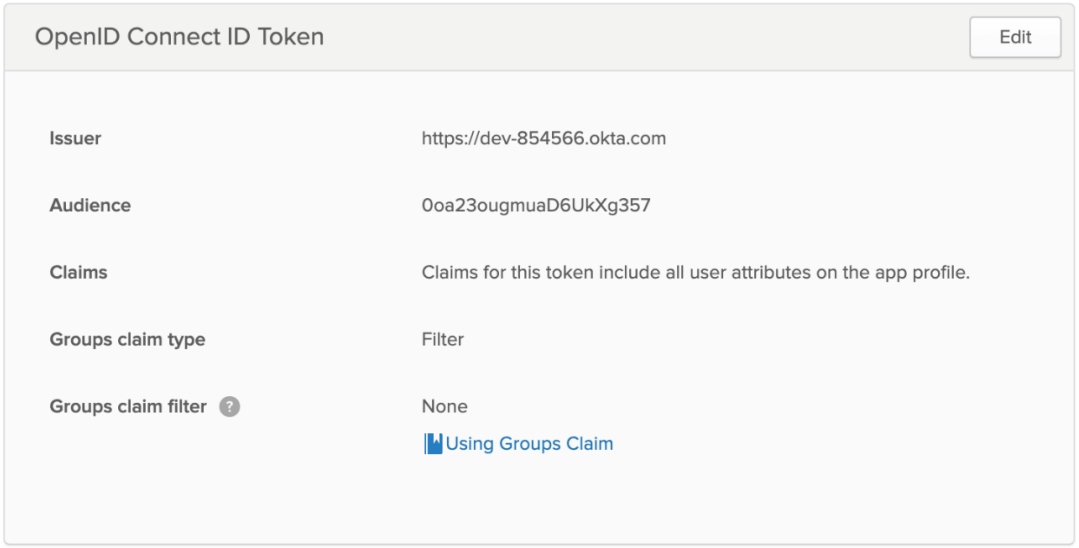

Abbildung 36: OKTA OpenID Connect Konfiguration

Weitere Konfigurationseinstellungen, betreffend der Claims des OpenID Connect Provider, beschränken sich auf die Standard-Einstellungen, da hierbei für die weitere Verwendung ausschließlich der *preferred_username* für die Identifikation relevant ist. Die Auswahl des Grant Type wurde gemäß OpenID Spezifikation auf den Authorization Code Flow (siehe Abschnitt 2.1.2) gesetzt. Als Redirect URIs, die per Intent unter Android aufgerufen werden, wurde *at.ac.fernfh.sechsm.app:/login* gesetzt, damit nach der Anmeldung über den System-Browser die App gestartet und der Authorization Code übergeben werden kann.

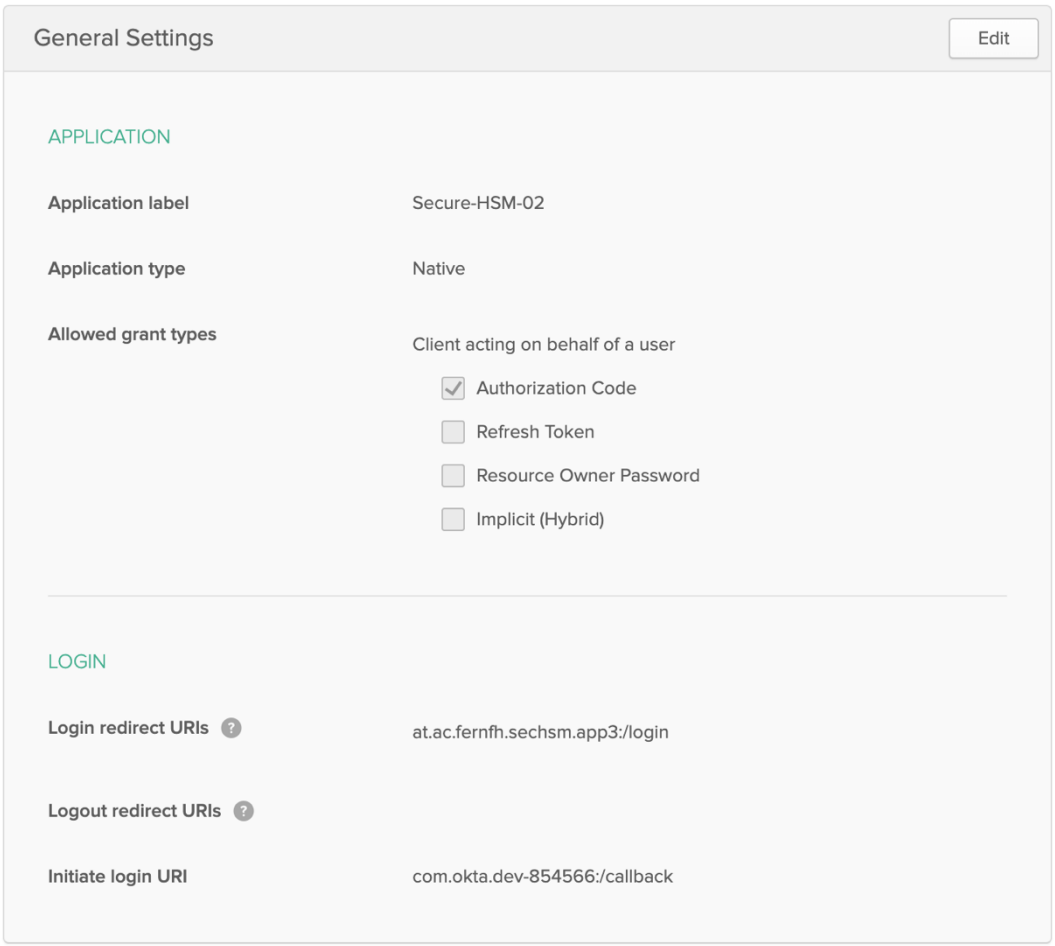

Abbildung 37: OpenID Connect Flow Konfiguration

Durch diese Konfiguration wird eine Authentifizierung gemäß der Empfehlung für native Apps des RFC 8252 [OA17], siehe Abschnitt 2.1.2.3, ermöglicht.

Um den Anforderungen von PKCE (Proof Key for Code Exchange) des RFC 7636 [Pr15], siehe Abschnitt 2.1.2.4, zu genügen, wird darüber hinaus diese Einstellung der *Client authentication* gesetzt. In der nachfolgenden Abbildung 38 ist diese Konfiguration des OpenID Connect Provider OKTA dargestellt.

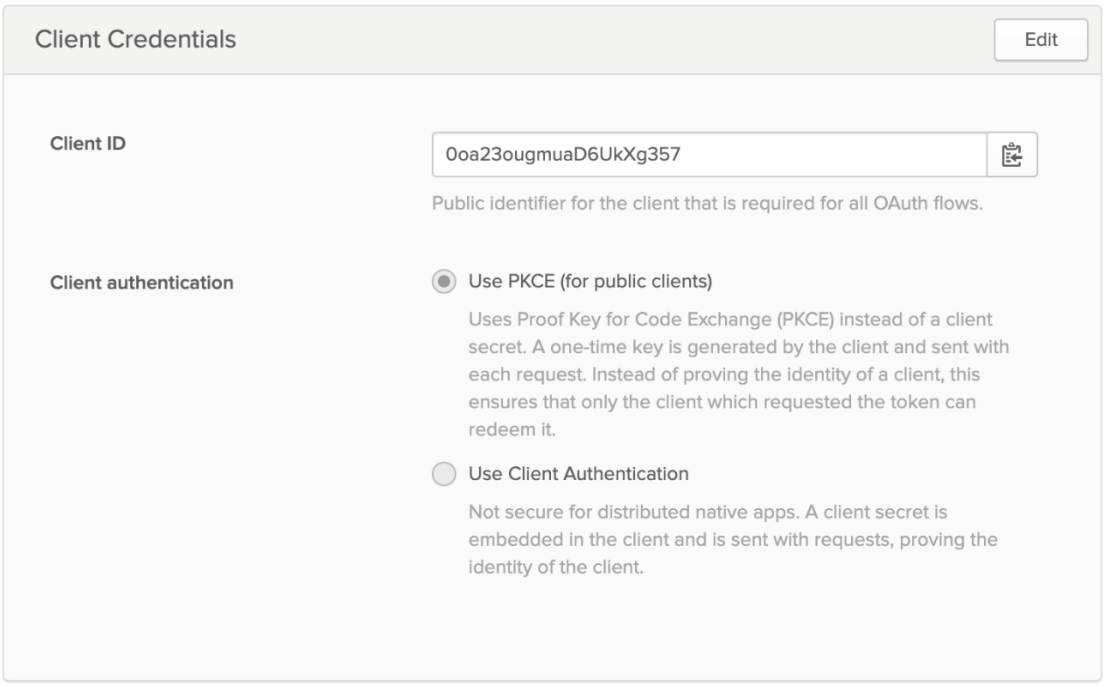

Abbildung 38: OpenID Connect Client Credentials

3.2.4.2 Umsetzung des mobilen Clients

Der mobile Client zur Demonstration wurde mit der Android Entwicklungsumgebung umgesetzt. Der Client verwendet zusätzlich ein Modul [ok20b] des OpenID Connect Providers OKTA, welches eine einfache Anmeldung über den Standard OpenID Connect ermöglicht.

Die umgesetzte mobile Anwendung bietet zur Überprüfung die in Tabelle 3 aufgelisteten bzw. in Abbildung 39 dargestellten Funktionen an.

Funktion	**Beschreibung**
Status	Anzeige des Status der Anwendung (Initialisierung).

Funktion	Beschreibung
Zurücksetzen	Abmeldung durchführen bzw. das Löschen zuvor generierter (TS) und importierter Schlüssel (DS).
Anmeldung	Durchführung der Anmeldung des Benutzers/der Benutzerin.
Registrierung	Initiierung der Registrierung der App.
Einrichtung	Abrufen des verschlüsselten DS und importieren dieses Schlüssels in die Secure Enclave.
Verwendung	Auflistung der verschlüsselten Daten zur Überprüfung der Verwendung es importierten Schlüssel (DS)

Tabelle 3: Funktionen der Secure-HSM App

Wesentliche Teile des Quellcodes der mobilen Anwendung sind im Anhang A ausgeführt. Dazu gehören:

1. Anmeldung über OpenID Connect durchführen.
2. Erstellung eines kryptographischen Schlüsselpaares.
3. Authentifizierung mittels Fingerabdruck durchführen.
4. Importieren des symmetrischen Schlüssels.

In der Abbildung 39 wird im linken Schnappschuss der App die Startseite dargestellt. Die Schaltflächen entsprechen den in Tabelle 3 beschriebenen Funktionen.

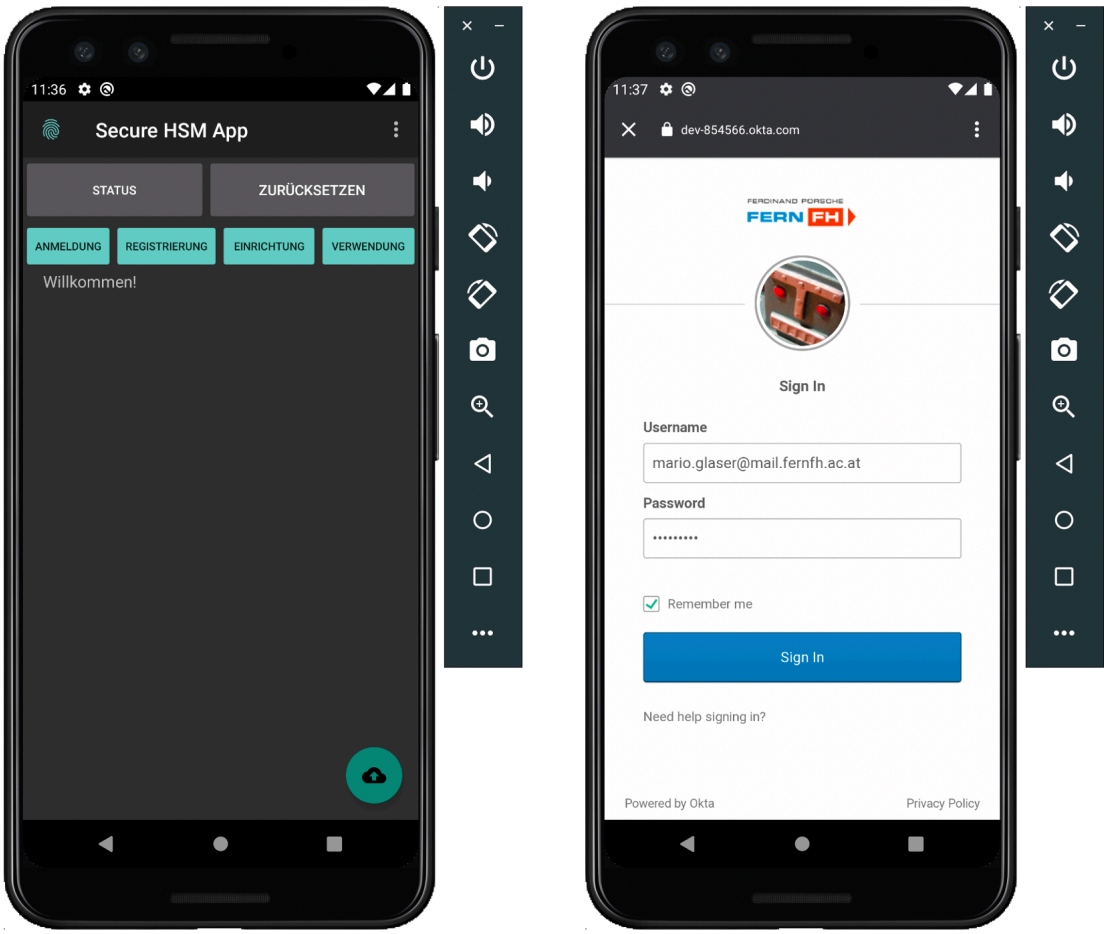

Abbildung 39: Startseite und Anmeldung der Secure-HSM-App

Im Zuge der Anmeldung öffnet sich der Browser des Geräts (siehe rechter Schnappschuss aus Abbildung 39) und die Anmeldeseite des OpenID Connect Providers OKTA wird dargestellt. Dabei müssen der Username und das Passwort eingegeben werden, sofern eine vorhergehende Anmeldung nicht weiter gültig ist. Der zughörige Quellecode dieser Routine ist in Anhang A *(Anmeldung über OpenID Connect durchführen)* gelistet.

Der Benutzer/die Benutzerin muss hierfür beim OpenID Connect Provider bereits registriert und für die Anwendung (Secure-HSM-02) berechtigt sein (siehe Abbildung 40).

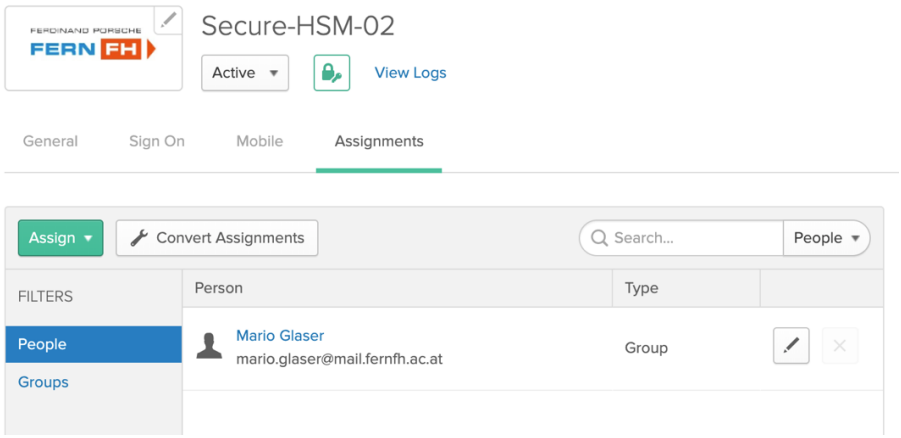

Abbildung 40: Zuordnung der Benutzer/Benutzerinnen für die Secure HSM App

Nach der Anmeldung führt der Browser ein Redirect (siehe Listing 12) auf die mobile Anwendung durch und übergibt den *Authorization Code* an diese.

```
1. dat=at.ac.fernfh.sechsm.app3:/login?
2. code=sEh88rPsWAYwg4mDZhZs&state=k1EsAE2WHkKdUfa2sOnAOA
```

Listing 12: Login Redirect (Remote Crypto Service)

Mit dem *Authorization Code* wird anschließend der *ID- und Access-Token* vom OpenID Connect Provider abgerufen und folgt damit dem Authorization Code Flow von OpenID Connect. Dieser ID-Token (siehe Listing 13) bestätigt die Identität des Benutzers/der Benutzerin und kann in weiterer Folge vom Backend Service des Prototypen für die Nutzung des Service genutzt werden. Nachfolgend ist der in Base 64 encodierte ID-Token des OpenID Connect Providers dargestellt.

```
1. eyJraWQiOiJWQ1ZVNjZrRm5iM1U0VlVYOG05R3F6dklkV3JvN3ZyeXh
2. qZjlOOFpmTTkwIiwiYWxnIjoiUlMyNTYifQ.eyJzdWIiOiIwMHV4enB
3. sMjZrSUZWMUVFbzM1NiIsIm5hbWUiOiJNYXJpbyBHbGFzZXIiLCJ2ZX
4. IiOjEsImlzcyI6Imh0dHBzOi8vZGV2LTg1NDU2Ni5va3RhLmNvbSIsI
5. mF1ZCI6IjBvYTIzb3VnbXVhRDZVa1hnMzU3IiwiaWF0IjoxNTgyOTMx
6. MjU0LCJleHAiOjE1ODI5MzQ4NTQsImp0aSI6IklELkdEQlY3TDc4TGd
7. zUzBzQzJJN2xDa2Zjcm9BZU43SGk3RVVSSENuYWxlWkkiLCJhbXIiOl
```

```
sicHdkIl0sImlkcCI6IjAwb3h6b3Q2M3dFa3ZGWHRMMzU2IiwibmuY
2UiOiJMU3pwVC1oazh4dFV1MU1zSF96OE5nIiwicHJlZmVycmVkX3Vz
ZXJuYW1lIjoibWFyaW8uZ2xhc2VyQG1haWwuZmVybmZoLmFjLmF0Iiw
iYXV0aF90aW1lIjoxNTgyOTMwOTQ1LCJhdF9oYXNoIjoib1htdm9hbF
JhSmFJQUtzdkFucTVWQSJ9.cZM3TpWJvycgYli_IdJIpzGfE78O-
bKSRFw4FWs1QrgYUK4kDLKzblUx3yilBZBywqUJx9pquynM_BC3qdwS
uSEO1b7XUSQLZEDGKAsDF6eQUHyVvwdM2ASKDU5dmQB8OgUakTXMDmS
fXvnlMZJS9DRUZneD0Kzk_al2OL4bPoSqpsaH_OJkU1WH_W0UlMRjjd
ymbt366rUIeCSf-cemsUhtgv8MAIMfIzpXp2r1wJb0RF0yNHBqqkeGy
pjE4ppf5ZvmgJZaA1pe6Ku3FdYGVAM0Vc1J_x7oglqnG3Swj8nq6unI
zzuVsU_fcqyNWWi9k2Eek5y8epbcWC7ccJpTJQ
```

Listing 13: ID Token (Remote Crypto Service)

Dieser *ID-Token* untergliedert sich in drei Bestandteile, die durch einen Punkt (.) getrennt sind. Der Erste Teil (siehe Listing 14) referenziert die Schlüssel ID *(kid)* und beschreibt den Algorithmus *(alg),* mit der die Signatur erstellt wurde:

```
{
  "kid": "VCVU66kFnb3U4VUX8m9GqzvIdWro7vryxjf9N8ZfM90",
  "alg": "RS256"
}
```

Listing 14: Teil 1 des ID-Tokens (Remote Crypto Service)

Beim Aufruf der aktuell eingesetzten Schlüssel des OpenID Connect Providers erhält man die Schlüssel bzw. Zertifikate zur Prüfung der Signatur über die referenzierte Schlüssel ID (*kid*).

Der Zweite Teil (siehe Listing 15 bzw. Tabelle 4) des ID-Tokens beinhaltet die nachfolgenden Attribute (*claims*):

```
{
  "sub": "00uxzpl26kIFV1EEo356",
```

```
3.   "name": "Mario Glaser",
4.   "ver": 1,
5.   "iss": "https://dev-854566.okta.com",
6.   "aud": "0oa23ougmuaD6UkXg357",
7.   "iat": 1582931254,
8.   "exp": 1582934854,
9.   "jti": "ID.GDBV7L78LgsS0sC2I7lCkfcroAeN7Hi7EURHCnaleZI",
10.  "amr": [
11.    "pwd"
12.  ],
13.  "idp": "00oxzot63wEkvFXtL356",
14.  "nonce": "LSzpT-hk8xtUu1MsH_z8Ng",
15.  "preferred_username":
16.    "mario.glaser@mail.fernfh.ac.at",
17.  "auth_time": 1582930945,
18.  "at_hash": "oXmvoalRaJaIAKsvAnq5VA"
19. }
```

Listing 15: Teil 2 des ID-Tokens

Attribut	Beschreibung
sub	ID des Benutzers/der Benutzerin, hier des Benutzers: Mario Glaser
name	Name des Benutzers/der Benutzerin.
aud	Client ID der mobilen Anwendung bzw. der Backend Komponenten.
iss	OpenID Connect Provider - Aussteller des Tokens.
iat	Ausstellungszeitpunkt des Tokens in Sekunden seit 1970.
exp	Ablaufdatum des Tokens in Sekunden seit 1970.

Attribut	Beschreibung
jti	Eindeutiger Identifier für den Authentifizierungs-Request.
amr	Authentifizierungsmethode des Benutzers/der Benutzerin, hier Passwort.
idp	ID des IdP (OpenID Connect Provider)
nonce	Eine Nonce wird eingesetzt, um Replay-Attacken zu verhindern.
preferred_username	User ID des Benutzers/der Benutzerin.
at_hash	Hash des OAuth 2.0 Access-Tokens.

Tabelle 4: Teil 2 des ID-Token (Remote Crypto Service)

Der dritte Teil des ID-Tokens ist für die Integrität und Authentizität des ID-Tokens zuständig. Durch die Signatur kann der Ressource Server überprüfen, ob diese von einem vertrauenswürdigen OpenID Connect Provider ausgestellt wurden.

3.2.5 Validierung

Die Überprüfung des Prototypen erfolgt über zwei verschiedene Abläufe. Für eine Identität (mario.a.glaser@mail.fernfh.ac.at) wird die *Secure-HSM-App* auf zwei unterschiedlichen Geräten installiert. Anschließend werden die Use-Cases Anmeldung, Registrierung, Einrichtung und Verwendung (siehe Abschnitt 3.1) durchgeführt und das Ergebnis dargestellt bzw. analysiert.

Folgende Testgeräte (siehe Abbildung 41) werden für die Validierung verwendet:

- Gerät 1: *Android Emulator für Google Pixel 3 (API Level 29, Build-Number QSR1.190920.001*
- Gerät 2: *Google Pixel 3 (API Level 29, Build-Number QQ2A.200501.001.B2)*

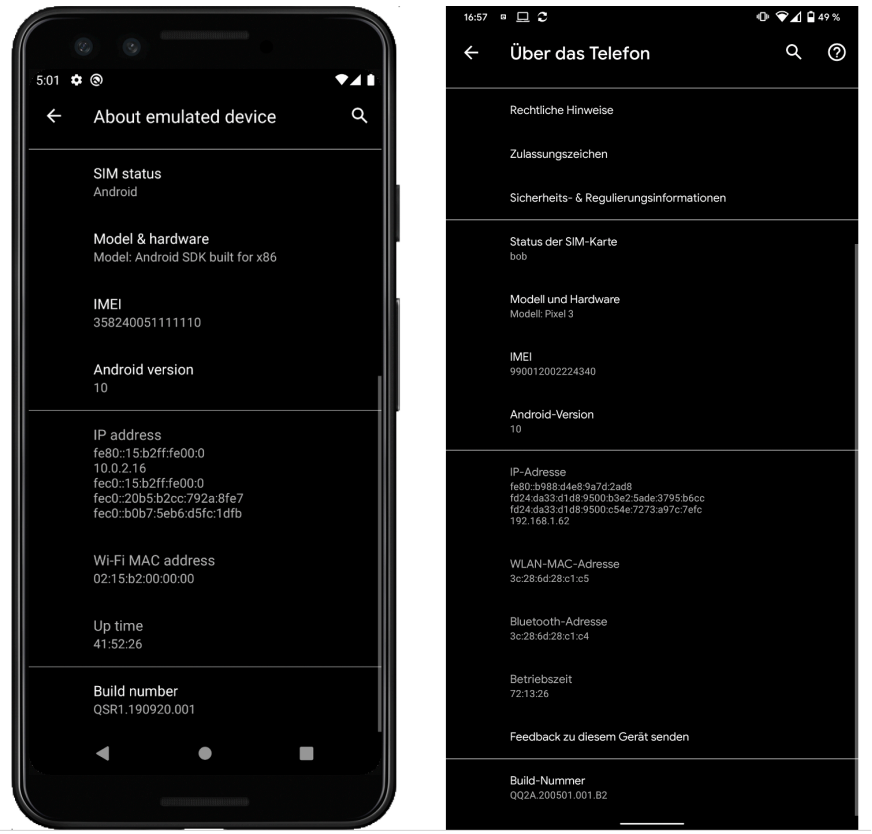

Abbildung 41: Gerät 1 und Gerät 2 der Validierung

Initial wird auf *Gerät 1* eine erstmalige Registrierung bzw. Einrichtung durchgeführt und somit am zentralen HSM der symmetrische Schlüssel (*DS*) zur Ver- und Entschlüsselung angelegt. Dieser Schlüssel wird anschließend auf dem Gerät 1 für das Hochladen von verschlüsselten Daten verwendet.

Im nächsten Schritt wird die Funktionstüchtigkeit des Prototypen auf zwei verschiedene Arten überprüft:

1. Im ersten Durchlauf wird die App auf dem Gerät 1 zurückgesetzt und somit werden alle Daten gelöscht. Bei erneuter Einrichtung der App mit derselben Identität werden die zuvor hochgeladenen Daten abgerufen.
2. Derselbe Durchlauf wird auf dem zweiten Gerät wiederholt und ein weiterer Datensatz hochgeladen.

Mit dieser Überprüfung wird gezeigt, dass auf dem mobilen *Gerät 1* und *Gerät 2* derselbe Schlüssel *DS* zur Ver- bzw. Entschlüsselung der Daten verwendet wird.

Vorbereitung

Die App wird auf dem *Gerät 1* eingerichtet und der Datensatz *„Upload Google Pixel 3 Emulator“* hochgeladen (siehe Abbildung 42). Beim Abruf der Daten werden diese im Secure Element entschlüsselt und in der App dargestellt. Dieser Datensatz wird im Testdurchlauf 1 und 2 für die Validierung herangezogen.

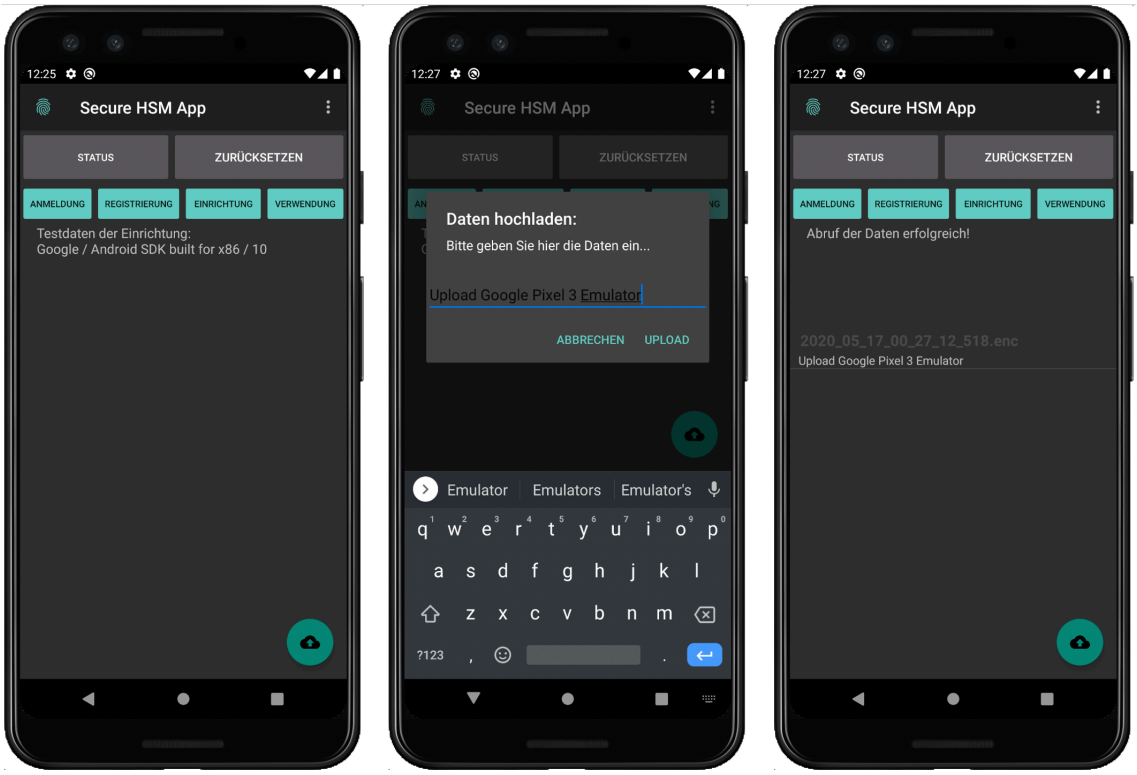

Abbildung 42: Vorbereitung der Validierung

Im nachfolgenden Auszug der LOG-Dateien der zentralen Anwendungen (siehe Listing 16) sind die wesentlichen Schritte der Durchführung aufgelistet. Über die Audience (**`00uxzpl26kIFV1EEo356`**) im ID-Token (siehe ebenfalls Listing 15 aus dem vorherigen Abschnitt 3.2.4) wird der Benutzer in weiterer Folge identifiziert. Für die *Registrierung* wird der Hash (SHA-1) über den öffentlichen Schlüssel

(*TS.pub*) berechnet (**9ad9a574419101a4462b88ea5194cb12128b4221**) und bei der *Einrichtung* im zentralen Schlüsselspeicher (siehe Listing 17) abgelegt. Im Zuge der Verwendung (Hochladen und Abrufen von Testdaten) werden für den Benutzer (**00uxzpl26kIFV1EEo356**) die Daten in seinem Verzeichnis abgelegt.

```
1.  Registrierung
2.  2020-05-17 00:24:48.050  INFO: Register public key for
3.  user mario.glaser@mail.fernfh.ac.at
4.  2020-05-17 00:24:48.265  INFO: Registered public key
5.  for user mario.glaser@mail.fernfh.ac.at with public
6.  key hash 9ad9a574419101a4462b88ea5194cb12128b4221
7.
8.  Einrichtung
9.  2020-05-17 00:25:35.526  INFO: Get encryption key for
10. <00uxzpl26kIFV1EEo356>
11. 2020-05-17 00:25:36.488  INFO: Install new public key
12. for user 00uxzpl26kIFV1EEo356 and public key hash
13. 9ad9a574419101a4462b88ea5194cb12128b4221
14. 2020-05-17 00:25:36.488  INFO: Create new Secret Key
15. for alias <00uxzpl26kIFV1EEo356>
16.
17. Hochladen des Testdatensatzes
18. 2020-05-17 00:27:14.323  INFO: Upload file for user
19. <00uxzpl26kIFV1EEo356> with name
20. <2020_05_17_00_27_12_518.enc>
21.
22. Abruf der Testdaten
23. 2020-05-17 00:27:21.315  INFO: List files for user
24. <00uxzpl26kIFV1EEo356>
25. 2020-05-17 00:27:21.316  INFO: Encrypted data for user
26. 00uxzpl26kIFV1EEo356 with filename
27. 2020_05_17_00_27_12_518.enc and data
```

```
28. SNomrg53ptn7feDYQMEkdRNiwmTHPQYZ6oQ3do3gnW2+eiRvG2Iry
29. 1W7NzgjqA==
```

Listing 16: Auszug der LOG-Dateien der Vorbereitung der Validierung

Der Auszug aus dem zentralen Schlüsselspeicher (**secret-key-store.jks**) und dem Verzeichnis (**00uxzpl26kIFV1EEo356**) der Ablage für den Benutzer (siehe Listing 17) zeigt, dass für den Benutzer der Schlüssel *DS* zur Ver- und Entschlüsselung der Daten und der öffentliche Schlüssel (*TS.pub*) für den Transport auf das mobile Gerät (in Form eines Zertifikats), sowie der verschlüsselte Datensatz gespeichert ist.

```
1.  Schlüsselspeicher
2.  a@b remote-crypto-key %
3.  keytool -list -keystore target/secret-key-store.jks
4.  -storepass 1234
5.  -storetype JCEKS
6.  Keystore-Typ: JCEKS
7.  Keystore-Provider: SunJCE
8.
9.  Keystore enthält 3 Einträge
10.
11. 00uxzpl26kifv1eeo356, 17.05.2020, SecretKeyEntry,
12. 00uxzpl26kifv1eeo356-9ad9a574419101a4462b88ea5194
13. cb12128b4221, 17.05.2020, trustedCertEntry,
14. Zertifikat-Fingerprint (SHA1):
15. 63:FF:C1:DF:62:D8:89:46:2A:10:EB:B0:1C:2E:B3:7F:09:
16. 1E:A9:55
17. ...
18.
19. Verzeichnis
20. a@b data-storage %
21. cat 00uxzpl26kIFV1EEo356/2020_05_17_00_27_12_518.enc
```

```
22.  SNomrg53ptn7feDYQMEkdRNiwmTHPQYZ6oQ3do3gnW2+eiRvG2Iry1W7Nz
23.  gjqA==%
```

Listing 17: Kryptographische Objekte und Daten nach der Vorbereitung der Validierung

Testdurchlauf 1

Die App (siehe Abbildung 43) auf dem *Google Pixel 3 Emulator* wird zurückgesetzt, um damit die Anmeldedaten und die kryptographischen Objekte im Secure Element zu löschen. Bei einer erneuten Einrichtung der App wird somit ein neues Schlüsselpaar *TS* im Secure Element erstellt und anschließend registriert.

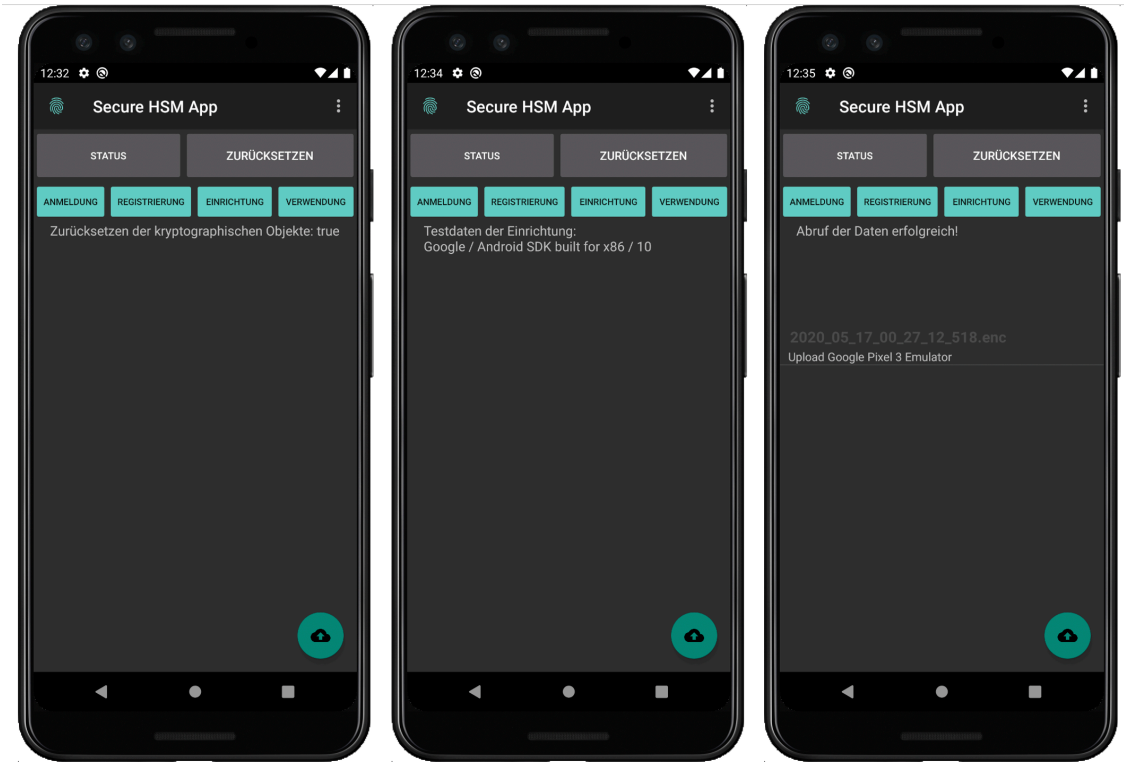

Abbildung 43: Abruf von Daten in Testdurchlauf 1

Der gekürzte Auszug der LOG-Dateien (siehe Listing 18) der zentralen Komponenten zeigt, dass für die Identität (**00uxzpl26kIFV1EEo356**) ein neuer öffentlicher Schlüssel *TS.pub* (**abd34444b5439ffc6a61ae33576dae92ae958d43**) registriert und bei der Einrichtung im zentralen Schlüsselspeicher abgelegt wird.

```
1.  Registrierung
2.  2020-05-17 00:32:18.256  INFO: Register public key for
3.  user mario.glaser@mail.fernfh.ac.at
4.  2020-05-17 00:32:18.261  INFO: Registered public key
5.  for user mario.glaser@mail.fernfh.ac.at with public
6.  key hash abd34444b5439ffc6a61ae33576dae92ae958d43
7.
8.  Einrichtung
9.  2020-05-17 00:34:10.002  INFO: Get encryption key for
10.  <00uxzpl26kIFV1EEo356>
11. 2020-05-17 00:34:10.015  INFO: Install new public key for
12.  user 00uxzpl26kIFV1EEo356 and public key hash
13. abd34444b5439ffc6a61ae33576dae92ae958d43
14. 2020-05-17 00:34:10.141  INFO: Alias already exists
15. <00uxzpl26kIFV1EEo356>
```

Listing 18: Auszug der LOG-Dateien des Testdurchlauf 1

Das Listing 19 zeigt, dass im zentralen Schlüsselspeicher für die Identität (**00uxzpl26kifv1eeo356**) ein weiterer öffentlicher Schlüssel *TS.pub* (**abd34444b5439ffc6a61ae33576dae92ae958d43**) hinzugefügt wurde.

```
1.  Schlüsselspeicher
2.  a@b remote-crypto-key %
3.  keytool -list -keystore target/secret-key-store.jks
4.  -storepass 1234
5.  -storetype JCEKS
6.  Keystore-Typ: JCEKS
7.  Keystore-Provider: SunJCE
8.
9.  Keystore enthält 4 Einträge
10.
11. 00uxzpl26kifv1eeo356, 17.05.2020, SecretKeyEntry,
```

```
00uxzpl26kifv1eeo356-9ad9a574419101a4462b88ea5194
cb12128b4221, 17.05.2020, trustedCertEntry,
Zertifikat-Fingerprint (SHA1):
63:FF:C1:DF:62:D8:89:46:2A:10:EB:B0:1C:2E:B3:7F:09:
1E:A9:55
00uxzpl26kifv1eeo356-abd34444b5439ffc6a61ae33576d
ae92ae958d43, 17.05.2020, trustedCertEntry,
Zertifikat-Fingerprint (SHA1):
EF:16:84:F3:4F:5B:71:25:E1:F7:E4:F1:81:83:0E:87:F1:
C3:01:35
...
```

Listing 19: Kryptographische Objekte nach Testdurchlauf 1

In der nachfolgenden Tabelle 5 sind die von der Komponente Secure-HSM-Key an die App zurücklieferten Ver- und Entschlüsselungsschlüssel *DS* aufgelistet. Damit wird gezeigt, dass unterschiedliche *TS.pub* bzw. am Endgerät *TS.priv* eingesetzt werden.

Vorbereitung	**Testdurchlauf 1**
68ef9e904713a3b5b1326da27a3	4010ce1accbff8a8497b72080fe
cf064fd324683f8c1e0358f38bc	d24982c6e13f7d6ea945a95fb8e
1de744c3cd80988a4ae074cb7d5	d5548d4740a9208a9e30c098b7a
e1b7dc5b87a7630ff6652507d36	35826f3cc442657908d94ec5b2a
88d4ccadf4e516ee3e31fb9ee16	fdff903253f712eeeb16f73e919
942867a90a6af5b10baebafdeab	5179a31634219772e5e4951dff7
...	...
febae663a7b6b8967a0ab556b7e	401dd199679a76b3d43cff259a3

Tabelle 5: Vergleich verschlüsselter DS Testdurchlauf 1

Testdurchlauf 2

Im zweiten Testdurchlauf wird überprüft, ob dieselben Daten *von Gerät 2* abgerufen und entschlüsselt werden können. Dabei werden die im *Testdurchlauf 1*

aufgelisteten Schritte erneut durchgeführt und anschließend die Daten abgerufen (siehe Abbildung 44) bzw. ein neuer Datensatz hochgeladen (siehe Abbildung 45).

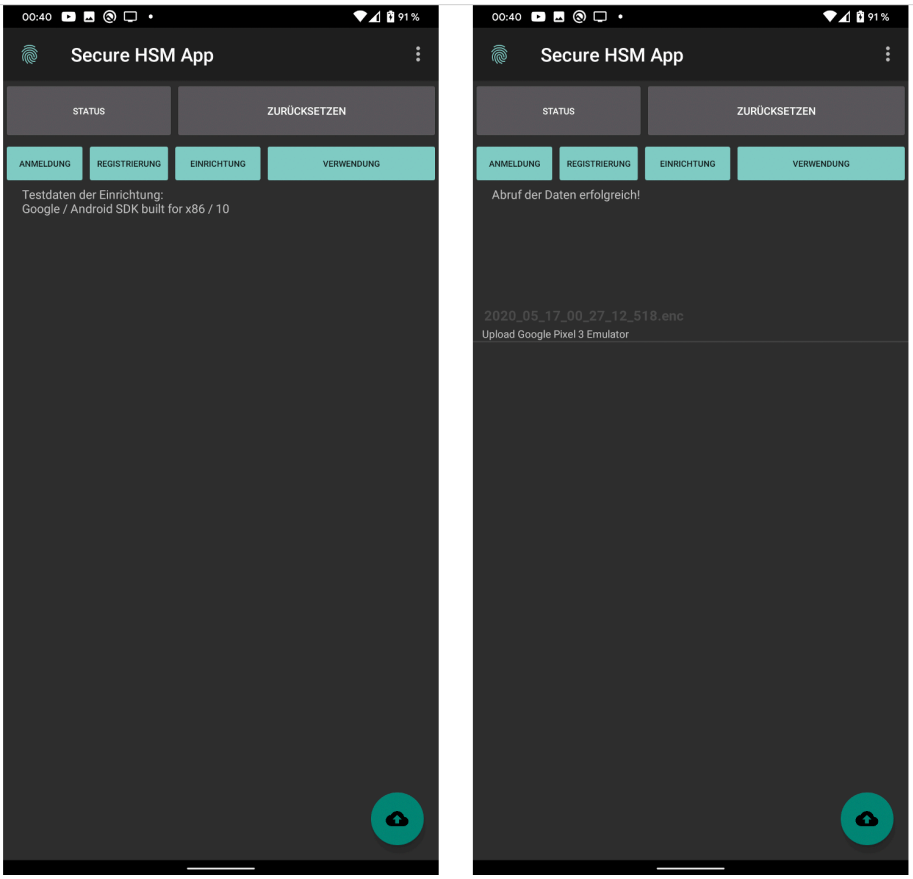

Abbildung 44: Abruf von Daten in Testdurchlauf 2

Aus den LOG-Dateien (Listing 20) der zentralen Komponenten ist im Testdurchlauf ersichtlich, dass ein weiterer *TS.pub* (**a7df7e2b784046b94e105589949d99a3c4c61823**) für die Identität (**00uxzpl26kIFV1EEo356**) registriert und eingerichtet wird.

```
1.  Registrierung
2.  2020-05-17 00:40:00.845  INFO: Register public key for
3.  user mario.glaser@mail.fernfh.ac.at
4.  2020-05-17 00:40:00.849  INFO: Registered public key
```

```
for user mario.glaser@mail.fernfh.ac.at with public key
hash a7df7e2b784046b94e105589949d99a3c4c61823

Einrichtung
2020-05-17 00:40:09.452  INFO: Get encryption key for
<00uxzpl26kIFV1EEo356>
2020-05-17 00:40:09.460  INFO:  Install new public key
for user 00uxzpl26kIFV1EEo356 and public key
hash a7df7e2b784046b94e105589949d99a3c4c61823
2020-05-17 00:40:09.569  INFO: Alias already exists
<00uxzpl26kIFV1EEo356>
```

Listing 20: Auszug der LOG-Dateien des Testdurchlauf 2

Der Auszug (Listing 20) zeigt, dass derselbe *DS* verwendet wird (**00uxzpl26kIFV1EEo356**). Da dieser allerdings mit einem anderen *TS.pub* verschlüsselt auf das Endgerät transportiert wird, ist der Wert des verschlüsselten *DS* somit ungleich (siehe Tabelle 6).

Vorbereitung	**Testdurchlauf 2**
68ef9e904713a3b5b1326da27a3 cf064fd324683f8c1e0358f38bc 1de744c3cd80988a4ae074cb7d5 e1b7dc5b87a7630ff6652507d36 88d4ccadf4e516ee3e31fb9ee16 942867a90a6af5b10baebafdeab ... febae663a7b6b8967a0ab556b7e	060afb1bbe996f304c1b91f8b9d a5cb782bda610e2dd24695a370c a1e9343706a080213795d44eec4 ccf7e71013e4fcb7009ea8172cc 9730b0e0e11d240c46e5c1c21cd ddb429aa291ec434fe2144a69b8 ... 953f520114c28c7ef001469c43c

Tabelle 6: Vergleich verschlüsselter DS Testdurchlauf 2

Abschließend wird auf Gerät 2 überprüft, ob der am mobilen Endgerät gespeicherte *DS* nicht nur zur Entschlüsselung, sondern auch zur Verschlüsselung für ein Hochladen von Daten verwendet werden kann. Dazu wird der Datensatz „Upload Google Pixel 3“ hochgeladen und anschließend beide Datenätze auf das Gerät 2 heruntergeladen.

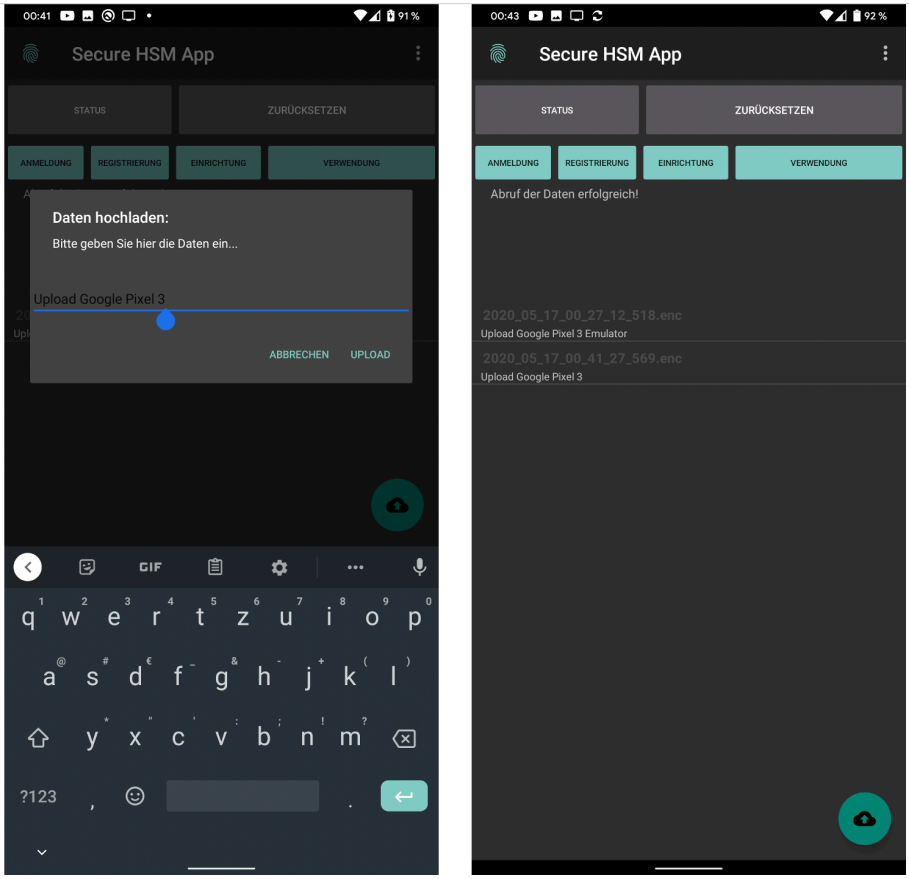

Abbildung 45: Hochladen von Datensatz in Testdurchlauf 2

Mit den beiden dargestellten Abläufen in Testdurchlauf 1 und 2 konnte somit erfolgreich gezeigt werden, dass der Prototyp auf mehreren mobilen Geräten korrekt funktioniert und Daten ver- bzw. entschlüsselt werden können.

4. Fazit

Zu Beginn dieses Kapitels wird eine Zusammenfassung der vorliegenden Arbeit gegeben. Anschließend wird das Ergebnis dargestellt und reflektiert. Zum Abschluss wird noch ein kurzer Ausblick auf mögliche Erweiterungen bzw. Fortsetzungen des *Remote Crypto Frameworks* aufgezeigt.

4.1 Zusammenfassung

In Kapitel 2 der Arbeit konnte die theoretische Grundlage geschaffen werden, um ein Framework zur Ver- und Entschlüsselung von Daten zu konzipieren und prototypisch umzusetzen. Dazu wurden SAML (Security Markup Language) und OpenID Connect als Vertreter für Protokolle zur Authentifizierung identifiziert und für den Einsatzbereich der mobilen Anwendungen analysiert. Wie in Abschnitt 2.4 dargestellt, bietet OpenID Connect ein geeignetes Protokoll zur Authentifizierung oberhalb des Frameworks OAuth 2.0. SAML 2.0 auf der anderen Seite ist ohne Adaption für den gewählten Einsatzbereich der mobilen Endgeräte nicht einsetzbar, da es, geschuldet auch durch seinen Entstehungszeitpunkt, moderne mobile Endgeräte nicht berücksichtigt. Ein weiterer identifizierter Vorteil von OpenID Connect ist die Unterstützung in der österreichischen Lösung E-ID und eine laufende Weiterentwicklung im Bereich der Herstellung einer starken Bindung der Anmeldung an das Gerät.

Das Framework zur Ver- und Entschlüsselung von Daten konnte anschließend in Kapitel 3 entworfen, modelliert und prototypisch umgesetzt werden. Ausgehend vom definierten Framework wurde anhand des konkreten Beispiels der Ver- und Entschlüsselung von Daten zur Dateiablage ein Service prototypisch implementiert. Durch die Implementierung einer Android App konnte gezeigt werden, dass kryptographische Objekte von einem zentralen Hardware Security Modul in den sicheren Speicher eines Smartphones transferiert werden können. Dazu wurden serverseitige Anwendungen implementiert, die das Schlüsselmaterial verschlüsselt an die App übermitteln, welche im Anschluss über die API die sichere

Entschlüsselung und Ablage im Secure Element ansteuert. Durch dieses Verfahren konnten im Anschluss Daten ver- und entschlüsselt werden, ohne dass der dafür verwendete Schlüssel in Klartext, außerhalb des zentralen Hardware Security Moduls oder dem Secure Element des mobilen Endgeräts, existiert.

4.2 Ergebnis

In der vorliegenden Arbeit konnte ein Framework konzipiert, dargestellt und prototypisch umgesetzt werden, dass unter Einsatz eines zentralen Schlüsselspeichers, eine sichere und vertrauensvolle Verwendung für kryptographische Schlüssel auf mobilen Endgeräten garantiert.

Im Zuge der Registrierung des Benutzers/der Benutzerin an dem zentralen Service zur Schlüsselverwaltung wird für diesen/diese ein spezifisches Schlüsselpaar in einem Hardware Security Modul generiert und an das mobile Endgerät übermittelt. Die Vertraulichkeit des Schlüsselpaares ist hierbei stets durch den Einsatz von kryptographischen Verfahren sichergestellt. So wird bei der Registrierung auf dem mobilen Endgerät ein Schlüsselpaar nach dem RSA Standard in der Secure Enclave generiert und bei der Registrierung wird der öffentliche Schlüssel des Benutzers/der Benutzerin zentral abgelegt. Der persönliche Schlüssel im zentralen Hardware Security Modul wird anschließend in verschlüsselter Form für das zuvor registrierte mobile Endgerät (öffentlicher Schlüssel) exportiert, um es anschließend in der Secure Enclave des mobilen Endgeräts wieder zu extrahieren und kann somit zu einem späteren Zeitpunkt verwendet werden.

Die initiale Authentifizierung des Benutzers/der Benutzerin wurde gemäß dem Standard OpenID Connect über den Cloud Anbieter OKTA realisiert. OpenID Connect bietet im Vergleich zu SAML (Security Markup Language) ohne Anpassung des Standards den Einsatz in nativen Anwendungen auf mobilen Endgeräten an.

Moderne mobile Endgeräte garantieren ein hohes Maß an Sicherheit. Zertifizierungen wie durch das National Institut of Standards and Technology

bestätigen, dass die dedizierten Sicherheitschips wie unter Apple iOS oder der Titan Chip von Google die Vertraulichkeit der kryptographischen Objekte garantieren.

Im der experimentellen Umsetzung konnte gezeigt werden, dass der Einsatz von Protokollen zur Authentifizierung, unter Wahrung der Sicherheit der Daten, möglich ist. Mit der Validierung des Prototypen kann somit die Forschungsfrage als gültig bestätigt werden: „*Es ist möglich ein Framework, für die Nutzung eines zentralen Hardware Security Modules zur Ver- und Entschlüsselung vom Smartphone, unter Berücksichtigung technischer Sicherheitsbedenken, basierend auf Standards im Bereich der Authentifizierung und Autorisierung, umzusetzen.*

Der Einsatz von starker Authentifizierung mit Unterstützung von kryptographischer Bindung ist aber weiterhin nur eingeschränkt gegeben. Basierend auf OpenID Connect bzw. OAuth 2.0 existieren bereits zwei erwähnenswerte RFCs (Request for Comments) für eine kryptographische Bindung des Access-Tokens an den Benutzer/die Benutzerin bzw. seine/ihre Authentifizierung, allerdings ist eine Verbreitung derzeit noch nicht gegeben.

4.3 Ausblick

In diesem Abschnitt wird ein Ausblick auf Weiterentwicklungen des *Remote Crypto Frameworks* gegeben.

4.3.1 Benutzerspezifisches „Geheimnis“

Die Verwendung des benutzerspezifischen Ver- und Entschlüsselungs-Schlüssels bzw. Schlüsselpaares des Benutzers/der Benutzerin, welcher im zentralen Schlüsselspeicher (HSM) abgelegt ist, beruht aktuell einzig auf der Integrität des Betreibers dieser zentralen Komponente. Zusätzlich zu diesem Schutz des Einsatzes eines HSM kann die Verwendung dieses sensiblen Schlüssels noch an einen zweiten Faktor wie das „Wissen“ des Benutzers/der Benutzerin, z.B. einer PIN geknüpft werden.

4.3.2 Token Binding

In den Abschnitten 2.1.2.5 und 2.1.2.6 wurden zwei Varianten einer Bindung von OpenID Connect Bearer Tokens an den Besitzer/die Besitzerin aufgezeigt. In beiden Lösungen werden diese Tokens kryptographisch an das lokale Gerät gebunden. Beide RFCs sind noch sehr neu und wurden daher bei der Umsetzung nicht berücksichtigt. Sie würden ein geeignetes Mittel darstellen, um das Entwenden der Tokens durch Angriffe zu unterbinden und können bei zukünftigen Arbeiten im Bereich der Authentifizierung über OpenID Connect berücksichtigt werden.

4.3.3 Authentifizierung über E-ID

Zum Zeitpunkt der Erstellung war die österreichische Lösung der E-ID gerade in der Umsetzung. Die gegenständliche Lösung dieser Arbeit basiert auf OpenID Connect und OKTA als OIDC Provider und bietet eine Grundlage für eine Erweiterung bzw. Anpassung der Authentifizierung über den Identity Provider von E-ID.

4.3.4 Kommunikation

Die Prüfung der Machbarkeit des Remote Crypto Frameworks (siehe Abschnitt 3.2) wurde anhand des Austauschs eines benutzerspezifischen symmetrischen Schlüssels durchgeführt. Dieses Verfahren könnte um den Austausch eines asymmetrischen Schlüssels erweitert werden, wie es z.B. für den verschlüsselten Austausch zwischen zwei Personen notwendig ist. Basierend auf dieser Erweiterung könnte der Einsatz in Messaging-Lösungen überprüft werden, um einen erweiterten Schutz der Daten vor Dritten zu gewähren.

4.3.5 Integration Fast Identity Online

Das Protokoll FIDO wurde in der Umsetzung des Prototypen (siehe Abschnitt 3.2) nicht berücksichtigt. FIDO bietet vergleichbar mit der Bindung der Tokens an das Gerät (siehe Abschnitt 4.3.1) ein Verfahren zur starken Bindung der Identität am Client an die Person an. Eine Integration von FIDO in die Abläufe der Authentifizierung erfordert eine Anpassung des Identity Providers, welcher in der

Machbarkeitsüberprüfung ausgeklammert wurde, jedoch für zukünftige Weiterentwicklungen auf Grundlage dieses Konzeptes und Prototypen integriert werden kann.

Literaturverzeichnis

[An20a] *Anbindung für Service Provider – Der österreichische E-ID*. URL https://eid.egiz.gv.at/secure/?page_id=259. - abgerufen am 2020-01-19

[An20b] *Android keystore system | Android-Entwickler*. URL https://developer.android.com/training/articles/keystore?hl=de. - abgerufen am 2020-02-01.

[Ap19] ATSEC INFORMATION SECURITY CORP: Apple Secure Key Store Cryptographic Module, v9.0 FIPS 140-2 Non-Proprietary Security Policy (2019)

[AR20] *ARCHITEKTUR – Der österreichische E-ID*. URL https://eid.egiz.gv.at/secure/?page_id=2. - abgerufen am 2020-02-21

[Bi05] CANTOR, SCOTT ; FREDERICK, HIRSCH ; KEMP, JOHN ; MALER, EVE: *Bindings for the oasis security assertion markup language (SAML) V2. 0*. URL https://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf

[DP18] DAMABI, HOSSEYNI ; PEDRAM, SEYED: Security analysis of the OpenID financial-grade API (2018)

[Ee20] *E-ID Roadmap – Der österreichische E-ID*. URL https://eid.egiz.gv.at/secure/?page_id=742. - abgerufen am 2020-01-19

[FI17] MACHANI, SALAH ; PHILPOTT, ROB ; SRINIVAS, SAMPATH ; KEMP, JOHN ; HODGES, JEFF: *FIDO UAF Architectural Overview*. URL https://fidoalliance.org/specs/fido-uaf-v1.1-ps-20170202/fido-uaf-overview-v1.1-ps-20170202.html. - abgerufen am 2020-01-22

[JS15] BRADLEY, JOHN ; SAKIMURA, NAT ; JONES, MICHAEL B.: *JSON Web Token (JWT)*. URL https://tools.ietf.org/html/rfc7519. - abgerufen am 2018-11-24

[Ke20] *KeyStore (Java Platform SE 8).* URL https://docs.oracle.com/javase/8/docs/api/java/security/KeyStore.html. - abgerufen am 2020-05-17

[LE14] LENZ, THOMAS ; ZWATTENDORFER, BERND ; STRANACHER, KLAUS ; TAUBER, ARNE: Identitätsmanagement in Österreich mit MOA-ID 2.0. In: *eGovernment review* Bd. 13 (2014), S. 18–19

[MSW16] MANDT, TARJEI ; SOLNIK, MATHEW ; WANG, DAVID: Demystifying the Secure Enclave Processor. In: , *OffCell Research*. (2016), S. 41

[NF11] NAVON, JAIME ; FERNANDEZ, FEDERICO: The Essence of REST Architectural Style. In: WILDE, E. ; PAUTASSO, C. (Hrsg.): *REST: From Research to Practice*. New York, NY : Springer New York, 2011 — ISBN 978-1-4419-8303-9, S. 21–33

[OA15] RICHER, JUSTIN: *OAuth 2.0 Token Introspection*. URL https://tools.ietf.org/html/rfc7662. - abgerufen am 2020-02-09

[OA17] BRADLEY, JOHN ; DENNISS, WILLIAM: *OAuth 2.0 for Native Apps*. URL https://tools.ietf.org/html/rfc8252. - abgerufen am 2020-01-05

[OA20] BRADLEY, JOHN ; CAMPBELL, BRIAN ; LODDERSTEDT, TORSTEN ; SAKIMURA, NAT: *OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens*. URL https://tools.ietf.org/html/rfc8705. - abgerufen am 2020-03-20

[Ok20a] *Okta | The Identity Standard*. URL https://www.okta.com/. - abgerufen am 2020-02-28

[ok20b] *okta/okta-oidc-android*. URL https://github.com/okta/okta-oidc-android. - abgerufen am 2020-02-29.

[Op14] SAKIMURA, N. ; BRADLEY, J. ; JONES, M ; DE MEDEIROS, B. ; MORTIMORE, C.: *OpenID Connect Core 1.0 incorporating errata set 1*. URL https://openid.net/specs/openid-connect-core-1_0.html#Authentication. - abgerufen am 2018-11-24.

[Pa17] PARECKI, A.: *OAuth 2.0 Simplified* : Lulu.com, 2017 — ISBN 978-1-387-30380-9

[PK15] GLEESON, SUSAN ; ZIMMAN, CHRIS: *PKCS #11 Cryptographic Token Interface Base Specification Version 2.40*. URL http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/os/pkcs11-base-v2.40-os.html. - abgerufen am 2020-04-01

[Pr05] HUGHES, JOHN ; CANTOR, SCOTT ; HODGES, JEFF ; FREDERICK, HIRSCH ; MISHRA, PRATEEK ; PHILPOTT, ROB ; MALER, EVE: *Profiles for the OASIS Security Assertion Markup Language (SAML) V2. 0*. URL http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf. - abgerufen am 2020-04-03

[Pr15] AGARWAL, NAVEEN ; SAKIMURA, NAT ; BRADLEY, J.: *Proof Key for Code Exchange by OAuth Public Clients*. URL https://tools.ietf.org/html/rfc7636. - abgerufen am 2020-01-05

[Sc16] SCHMEH, KLAUS: *Kryptografie: Verfahren, Protokolle, Infrastrukturen, iX-Edition*. 6., aktualisierte Auflage. Heidelberg : dpunkt.verlag, 2016 — ISBN 978-3-86490-356-4

[Se08] RAGOUZIS, NICK ; HUGHES, JOHN ; PHILPOTT, ROB ; MALER, EVE ; MADSEN, PAUL ; SCAVO, TOM: *Security Assertion Markup Language (SAML) V2.0 Technical Overview*. URL http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0-cd-02.html. - abgerufen am 2020-02-07

[SHB15] SCHREINER, MICHEL ; HESS, THOMAS ; BENLIAN, ALEXANDER: *Gestaltungsorientierter Kern oder Tendenz zur Empirie? Zur neueren methodischen*

Entwicklung der Wirtschaftsinformatik (Working Paper Nr. 1/2015) : Arbeitsbericht, Institut für Wirtschaftsinformatik und Neue Medien, Fakultät für Betriebswirtschaft, Ludwig-Maximilians-Universität, 2015

[SM18a] SCHWARTZ, MICHAEL ; MACHULAK, MACIEJ: OAuth. In: *Securing the Perimeter: Deploying Identity and Access Management with Free Open Source Software*. Berkeley, CA : Apress, 2018 — ISBN 978-1-4842-2601-8, S. 105–149

[SM18b] SCHWARTZ, MICHAEL ; MACHULAK, MACIEJ: Strong Authentication. In: *Securing the Perimeter: Deploying Identity and Access Management with Free Open Source Software*. Berkeley, CA : Apress, 2018 — ISBN 978-1-4842-2601-8, S. 231–265

[SM18c] SCHWARTZ, MICHAEL ; MACHULAK, MACIEJ: OpenID Connect. In: *Securing the Perimeter: Deploying Identity and Access Management with Free Open Source Software*. Berkeley, CA : Apress, 2018 — ISBN 978-1-4842-2601-8, S. 151–203

[Sr14] SIRIWARDENA, PRABATH: OpenID Connect. In: SIRIWARDENA, P. (Hrsg.): *Advanced API Security: Securing APIs with OAuth 2.0, OpenID Connect, JWS, and JWE*. Berkeley, CA : Apress, 2014 — ISBN 978-1-4302-6817-8, S. 181–200

[Sr20a] SIRIWARDENA, PRABATH: Accessing APIs via Native Mobile Apps. In: *Advanced API Security: OAuth 2.0 and Beyond*. Berkeley, CA : Apress, 2020 — ISBN 978-1-4842-2050-4, S. 227–241

[Sr20b] SIRIWARDENA, PRABATH: OAuth 2.0 Security. In: *Advanced API Security: OAuth 2.0 and Beyond*. Berkeley, CA : Apress, 2020 — ISBN 978-1-4842-2050-4, S. 287–304

[Sr20c] SIRIWARDENA, PRABATH: OAuth 2.0 Token Binding. In: *Advanced API Security: OAuth 2.0 and Beyond*. Berkeley, CA : Apress, 2020 — ISBN 978-1-4842-2050-4, S. 243–255

[Sp20a] *Specifications – OpenID*. URL https://openid.net/developers/specs/. - abgerufen am 2018-11-25

[Sp20b] *Spring Boot*. URL https://spring.io/projects/spring-boot. - abgerufen am 2020-02-28.

[Th10] HAMMER-LAHAV <ERAN@HUENIVERSE.COM>, ERAN: *The OAuth 1.0 Protocol*. URL https://tools.ietf.org/html/rfc5849. - abgerufen am 2020-02-09

[Th12] HARDT <DICK.HARDT@GMAIL.COM>, DICK: *The OAuth 2.0 Authorization Framework*. URL https://tools.ietf.org/html/rfc6749. - abgerufen am 2018-11-24

[Th18b] BALFANZ, DIRK ; HODGES, JEFF ; POPOV, ANDREY: *The Token Binding Protocol Version 1.0*. URL https://tools.ietf.org/html/rfc8471. - abgerufen am 2020-01-16

[TJ14] VAN TILBORG, H.C.A. ; JAJODIA, S.: *Encyclopedia of Cryptography and Security*, *Encyclopedia of Cryptography and Security* : Springer US, 2014 — ISBN 978-1-4419-5906-5

[Tr18] BALFANZ, DIRK ; POPOV, ANDREY: *Transport Layer Security (TLS) Extension for Token Binding Protocol Negotiation*. URL https://tools.ietf.org/html/rfc8472. - abgerufen am 2020-03-20

[TH19] THEUERMANN, KEVIN ; ZEFFERER, THOMAS ; LENZ, THOMAS ; TAUBER, ARNE: Flexible und benutzerfreundliche Authentifizierungsverfahren zur Umsetzung transaktionaler E-Government-Services auf mobilen Geräten. In: STEMBER, J. ; EIXELSBERGER, W. ; SPICHIGER, A. ; NEURONI, A. ; HABBEL, F.-R. ; WUNDARA, M. (Hrsg.): *Handbuch E-Government: Technikinduzierte Verwaltungsentwicklung*. Wiesbaden : Springer Fachmedien Wiesbaden, 2019 — ISBN 978-3-658-21402-9, S. 405–434

[Ti18] *Titan M makes Pixel 3 our most secure phone yet*. URL https://blog.google/products/pixel/titan-m-makes-pixel-3-our-most-secure-phone-yet/. - abgerufen am 2020-02-24.

[ZTL11] ZEFFERER, THOMAS ; TEUFL, PETER ; LEITOLD, HERBERT: Mobile qualifizierte Signaturen in Europa. In: *Datenschutz und Datensicherheit - DuD* Bd. 35 (2011), Nr. 11, S. 768–773

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

AD	Active Directory
AES	Advanced Encryption Standard
API	Application Programming Interface
CPU	Central Processing Unit
DS	Daten-Schlüssel bzw. Schlüsselpaar
EGIZ	E-Government Innovationszentrum
E-ID	Elektronische Identität
FIDO	Fast Identity Online
GUI	Graphical User Interface
HSM	Hardware Security Modul
HTTP	Hyper Text Transfer Protokoll

IDP	Identity Provider
JOSE	JavaScript Object Signing and Encryption
JWT	JSON Web Token
JSON	JavaScript Object Notation
MOA-ID	Modul für Online Applikationen - Identität
NIST	National Institute of Standards and Technology
OASIS	Organization for the Advancement of Structured Information Standards
OIDC	OpenID Connect
PIN	Persönliche Identifikationsnummer
PKI	Public Key Infrastruktur
PKCE	Proof Key for Code Exchange
PKCS	Public Key Cryptography Standards
PRIV	Private Key (Privater Schlüssel)
PUB	Public Key (Öffentlicher Schlüssel)
REST	Representational State Transfer
RFC	Request For Comment
RSA	RSA-Kryptosystem nach Rivest, Shamir und Adleman
SAML	Security Markup Language
SE	Secure Element
SEP	Secure Enclave Processor
SMS	Short Message Service

SHA	Secure Hash Algorithm
SOAP	Simple Object Access Protocol
SOC	System On a Chip
SP	Service Provider
SSEE	Sichere Signaturerstellungseinheit
SSO	Single Sign On
TAN	Transaktionsnummer
TEE	Trusted Execution Environment
TLS	Transport Layer Security
TS	Transport Schlüsselpaar
URL	Uniform Resource Locator
XML	Extensible Markup Language

ANHANG A

In diesem Anhang werden relevante Ausschnitte des Quellcodes der mobilen App aufgelistet. Der vollständige Quellcode ist in der beigefügten CD enthalten.

Anmeldung über OpenID Connect durchführen

Nachfolgend (siehe Listing 21) ist ein Auszug der OpenID Connect Konfiguration in der mobilen Anwendung bzw. die zughörige Konfigurationsdatei (siehe Listing 22).

```
1.  // Initialisiere OpenID Connect Konfiguration fuer OKTA
2.  OIDCConfig config = new OIDCConfig.Builder()
3.          .withJsonFile(this, R.raw.okta_oidc_config)
4.          .create();
5.
6.  mWebAuth = new Okta.WebAuthBuilder()
7.          .withConfig(config)
8.          .withContext(getApplicationContext())
9.          .withStorage(new SharedPreferenceStorage(this))
10.         .withCallbackExecutor(Executors.newSingleThreadExecutor())
11.         .withTabColor(Color.BLUE)
12.         .setRequireHardwareBackedKeyStore(false)
13.         .supportedBrowsers("com.android.chrome", "org.mozilla.firefox")
14.         .create();
15.
16. mSessionClient = mWebAuth.getSessionClient();
17.
18. ResultCallback<AuthorizationStatus, AuthorizationException> callback =
19.         new ResultCallback<AuthorizationStatus, AuthorizationException>() {
20.             @Override
21.             public void onSuccess(@NonNull AuthorizationStatus status) {
22.                 Log.i("AUTHORIZED", "Success");
23.                 if (status == AuthorizationStatus.AUTHORIZED) {
24.                     //client is authorized.
25.                     try {
26.                         Tokens tokens = mSessionClient.getTokens();
27.                         TextView textView = findViewById(R.id.textView);
28.
29.                         //textView.setText("Login successful\n");
30.
31.                     } catch (AuthorizationException e) {
```

```
                            Log.e("AUTH", "Error get tokens from session", e);
                        }
                    } else if (status == AuthorizationStatus.SIGNED_OUT) {
                        //this only clears the browser session.

                        TextView textView = findViewById(R.id.textView);
                        textView.setText("Abmeldung erfolgreich\n");
                    }
                }

                @Override
                public void onCancel() {

                    Log.i("AUTHORIZED", "Cancel");
                    TextView textView = findViewById(R.id.textView);
                    textView.setText("Anmeldung abgebrochen\n");
                }

                @Override
                public void onError(@Nullable String msg, AuthorizationException error)
                {
                    Log.i("AUTHORIZED", "Error");
                    TextView textView = findViewById(R.id.textView);
                    textView.setText("Fehler bei der Anmeldung\n");
                }
            };
mWebAuth.registerCallback(callback, this);
```

Listing 21: Datei MainActivity.java

```
{
  "client_id": "0oa23ougmuaD6UkXg357",
  "redirect_uri": "at.ac.fernfh.sechsm.app3:/login",
  "end_session_redirect_uri": "at.ac.fernfh.sechsm.appX:/logout",
  "scopes": [
    "openid",
    "profile",
    "offline_access"
  ],
  "discovery_uri": "https://dev-854566.okta.com/"
}
```

Listing 22: Datei okta_oidc_config.json

Erstellung eines kryptographischen Schlüsselpaares

Nachfolgende Auflistung (Listing 23) zeigt wie im Secure Element des mobilen Endgeräts ein lokales Schlüsselpaar generiert wird. Dieses wird in weiterer Folge für die *Registrierung* und den *Import* des Verschlüsselungsschlüssel in das Secure Element genutzt.

```
1.  private void generateKeyPair()
2.          throws GeneralSecurityException, IOException {
3.      KeyStore ks = getKeyStore();
4.
5.      KeyPairGenerator kpg = KeyPairGenerator.getInstance(
6.              KeyProperties.KEY_ALGORITHM_RSA, "AndroidKeyStore");
7.      KeyGenParameterSpec spec = new KeyGenParameterSpec.Builder(
8.              alias,
9.              KeyProperties.PURPOSE_SIGN | KeyProperties.PURPOSE_VERIFY |
10.             KeyProperties.PURPOSE_WRAP_KEY | KeyProperties.PURPOSE_ENCRYPT |
11.             KeyProperties.PURPOSE_DECRYPT)
12.             .setDigests(KeyProperties.DIGEST_SHA256, KeyProperties.DIGEST_SHA512)
13.             .setBlockModes(KeyProperties.BLOCK_MODE_ECB,
14.             KeyProperties.BLOCK_MODE_ECB, KeyProperties.BLOCK_MODE_CTR,
15.             KeyProperties.BLOCK_MODE_GCM)
16.             .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_RSA_PKCS1)
17.             .setUserAuthenticationRequired(true)
18.             .setUserAuthenticationValidityDurationSeconds(30)
19.             .setKeySize(2048).build();
20.
21.     kpg.initialize(spec);
22.
23.     mPair = kpg.generateKeyPair();
24.
25.     unwrapCipher.init(Cipher.UNWRAP_MODE, mPair.getPrivate());
26. }
```

Listing 23: Datei CryptoService.java

Authentifizierung mittels Fingerabdruck durchführen

Nachfolgender Auflistung (Listing 24) zeigt den Auszug aus der *Activity* der Android Anwendung, welche nach erfolgreicher Authentifizierung des Benutzers den Zugriff

auf das Secure Element freigibt und anschließend den „wrapped“ Verschlüsselungs-Schlüssel importiert und eine Entschlüsselung durchführt.

```
1.  private void showBiometricPrompt() {
2.      BiometricPrompt.PromptInfo promptInfo =
3.              new BiometricPrompt.PromptInfo.Builder()
4.                      .setTitle("Authentifizierung mittels Biometrie")
5.                      .setSubtitle("Anmeldung durchfuehren")
6.                      .setNegativeButtonText("Abbrechen")
7.                      .build();
8.
9.      BiometricPrompt biometricPrompt = new BiometricPrompt(MainActivity.this,
10.             executor, new BiometricPrompt.AuthenticationCallback() {
11.         @Override
12.         public void onAuthenticationError(int errorCode,
13.                                           @NonNull CharSequence errString) {
14.             super.onAuthenticationError(errorCode, errString);
15.             Toast.makeText(getApplicationContext(),
16.                 "Fehler bei der Authentifizierung:" + errString,
17.                  Toast.LENGTH_SHORT)
18.                  .show();
19.         }
20.
21.         @Override
22.         public void onAuthenticationSucceeded(
23.                 @NonNull BiometricPrompt.AuthenticationResult result) {
24.             super.onAuthenticationSucceeded(result);
25.
26.             BiometricPrompt.CryptoObject authenticatedCryptoObject =
27.                     result.getCryptoObject();
28.
29.
30.             String decryptedData = callKeyService();
31.             TextView textView = findViewById(R.id.textView);
32.
33.             textView.setText("Decrypted Data: \n" + decryptedData);
34.         }
35.
36.         @Override
37.         public void onAuthenticationFailed() {
38.             super.onAuthenticationFailed();
39.             Toast.makeText(getApplicationContext(), "Authentifizierung
40.               fehlerhaft",
```

```
41.                     Toast.LENGTH_SHORT)
42.                     .show();
43.         }
44.     });
45.
46.     BiometricPrompt.CryptoObject cipher = new
47.        BiometricPrompt.CryptoObject(cryptoService.getUnwrapCipher());
48.
49.     biometricPrompt.authenticate(promptInfo, cipher); //, cipher);
50. }
```

Listing 24: Datei MainActivity.java

Importieren des symmetrischen Schlüssels

Im nachfolgenden Auszug (Listing 25) ist aufgelistet, wie der „wrapped" Verschlüsselungsschlüssel in das Secure Element am mobilen Endgerät importiert wird.

```
1.  public void unwrap(byte[] wrappedKey) throws GeneralSecurityException, IOException
2.  {
3.    Log.i("UNWRAP", "Start unrapping the secret key");
4.    if (checkEncryptionKey()) {
5.      throw new IllegalStateException("Secret key is already available - call
6.                                      reset before unwrap key");
7.    }
8.
9.    KeyStore keyStore = getKeyStore();
10. // unwrap it again with the RSA key from the keystore
11.   SecretKey secretKey = unwrapInternal(wrappedKey);
12.
13.   Log.i("TAG", "Set Secret Key Entry");
14.   KeyStore.SecretKeyEntry secretKeyEntry = new KeyStore.SecretKeyEntry(secretKey);
15.   keyStore.setEntry(getEncryptionKeyAlias(), secretKeyEntry,
16.     new KeyProtection.Builder(KeyProperties.PURPOSE_ENCRYPT +
17.                               KeyProperties.PURPOSE_DECRYPT)
18.     .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_NONE)
19.     .setUserAuthenticationValidityDurationSeconds(300)
20.     .setRandomizedEncryptionRequired(false)
21.     .setBlockModes(KeyProperties.BLOCK_MODE_CBC, KeyProperties.BLOCK_MODE_CBC,
22.                    KeyProperties.BLOCK_MODE_GCM)
23.     .setUserAuthenticationRequired(false)
```

```
24.     .build());
25.
26.   Log.i("TAG", "Secret Key imported");
27. }
```

Listing 25: Datei CryptoService.java

ANHANG B

In diesem Anhang sind die relevanten Quellcode Ausschnitte der Backend Komponenten aufgelistet.

Konfiguration der Anwendung als OpenID Connect Ressource Server

Die nachfolgende Klasse (Listing 26) ist über alle drei Komponenten ident. Alle Komponenten nutzen die idente Konfiguration als OpenID Connect Ressource Server.

```
1.  /**
2.   * OpenID bzw. OAuth 2.0 Konfiguration.
3.   *
4.   * @author Mario Glaser
5.   * @since 1.0
6.   */
7.  @EnableWebSecurity
8.  public class SecurityConfig {
9.
10.     @Configuration
11.     static class OktaOAuth2WebSecurityConfigurerAdapter extends
12.         WebSecurityConfigurerAdapter {
13.
14.         @Override
15.         protected void configure(HttpSecurity http) throws Exception {
16.             // @formatter:off
17.             http
18.                     .authorizeRequests().anyRequest().authenticated()
19.                     .and()
20.                     .oauth2Login().and()
21.                     .oauth2ResourceServer().jwt().jwkSetUri(
22.                          "https://dev-854566.okta.com/oauth2/v1/keys");
23.             // @formatter:on
24.         }
25.     }
26.
27.     @Bean
28.     ClientRegistrationRepository clientRegistrationRepository() {
29.         ClientRegistration clientRegistration = ClientRegistrations
30.                 .fromOidcIssuerLocation("https://dev-854566.okta.com")
```

```
31.                 .clientId("0oa23ougmuaD6UkXg357")
32.                 .build();
33.
34.         return new InMemoryClientRegistrationRepository(clientRegistration);
35.     }
36. }
```

Listing 26: SecurityConfig.java

Registrieren eines Benutzers

Der nachfolgende Auszug zeigt die Registrierung eines Benutzers auf Grundlage seines öffentlichen Schlüssels. Dieser Schlüssel wurde am Client (App) generiert und wird bei der Registrierung übermittelt (siehe Listing 27).

```
1.  @Service
2.  @Slf4j
3.  public class RegistrationService {
4.
5.    @Autowired
6.    private UserRepository userRepository;
7.
8.    @Transactional
9.    public void registerAppForUser(Registration registration, JwtAuthenticationToken
10.     jwtAuthenticationToken) throws GeneralSecurityException {
11.     log.info("Register user {}", registration);
12.     PublicKey publicKey = SecHSMUtil.parsePublicKey(registration.getPublicKey());
13.
14.     log.info("Register public key for user {}",
15.     jwtAuthenticationToken.getToken().getClaimAsString("preferred_username"));
16.     RegisteredUser user = new RegisteredUser();
17.     user.setIdToken(jwtAuthenticationToken.getToken().getTokenValue());
18.     user.setEMail(
19.        jwtAuthenticationToken.getToken().getClaimAsString("preferred_username"));
20.     user.setAuthTime(
21.        jwtAuthenticationToken.getToken().getClaimAsInstant("auth_time"));
22.     user.setExpTime(jwtAuthenticationToken.getToken().getClaimAsInstant("exp"));
23.     user.setSubject(jwtAuthenticationToken.getToken().getSubject());
24.     user.setPublicKey(SecHSMUtil.parsePublicKey(registration.getPublicKey()));
25.     user.setPulbicKeyValue(Base64.getEncoder().encodeToString(
26.         registration.getPublicKey()));
27.     user.setPulbicKeyHash(Hex.toHexString(
```

```
28.         SecHSMUtil.getPublicKeyHash(publicKey)));
29.     user.setDeviceInfo(registration.getDeviceInfo());
30.
31.     userRepository.save(user);
32.     log.info("Registered public key for user {} with public key hash {}",
33.       user.getEMail(),
34.       user.getPulbicKeyHash());
35.   }
36. }
```

Listing 27: Registrieren eines Benutzers

Zugriff auf den Schlüsselspeicher

Der Zugriff auf das Hardware Security Modul (HSM) wird über einen Java Key Store simuliert.

```
1. @Slf4j
2. @Component
3. public final class SoftwareCryptoProvider implements KeyCryptoProvider {
4.
5. //...gekürzt.
6.
7.   @Override
8.   public void installPublicKey(String alias, PublicKey publicKey, String testData)
9.     throws GeneralSecurityException {
10.
11.    SecretKey secretKey = (SecretKey) secretKeyStore.getKey(alias,
12.      configuration.getPassword().toCharArray());
13.
14.    if (secretKey != null) {
15.      log.info("Alias already exists <{}>", alias);
16.    } else {
17.      log.info("Create new Secret Key for alias <{}>", alias);
18.      secretKey = createSecretKey();
19.
20.      secretKeyStore.setEntry(alias, new SecretKeyEntry(secretKey),
21.      new KeyStore.PasswordProtection(configuration.getPassword().toCharArray()));
22.
23.      store();
24.    }
25.
26.    String publicKeyHash =
```

```
27.       Hex.toHexString(SecHSMUtil.getPublicKeyHash(publicKey));
28.
29.     final String userAlias = alias + "-" + publicKeyHash;
30.
31.     if (!secretKeyStore.containsAlias(userAlias)) {
32.       final PrivateKey privateKey = (PrivateKey)
33.           secretKeyStore.getKey(REMOTE_CRYPTO_CA_ALIAS,
34.           configuration.getPassword().toCharArray());
35.       final X509Certificate userCertificate = generateUserCertificate(privateKey,
36.           publicKey, alias);
37.       secretKeyStore.setCertificateEntry(userAlias, userCertificate);
38.       store();
39.     }
40.
41.     final byte[] wrappedSecretKey;
42.     final byte[] encryptedTestData;
43.     try {
44.       wrappedSecretKey = wrapSecretKey(
45.         secretKeyStore.getCertificate(userAlias).getPublicKey(),
46.         secretKey);
47.       encryptedTestData = getEncrypteTestData(secretKey, testData);
48.     } catch (GeneralSecurityException e) {
49.       throw new GeneralSecurityException("Could not create the wrapping key", e);
50.     }
51.
52.     keyMap.put(alias, new Triple(publicKey, wrappedSecretKey, encryptedTestData));
53.   }
```

Listing 28: Datei: SoftwareCryptoProvider.java