Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik

Biometriebasierte Authentifizierung mit WebAuthn

Masterarbeit

zur Erlangung des akademischen Grades
Master of Science (M. Sc.)

eingereicht von: Malte Kruse
geboren am:
geboren in:

Gutachter/innen: Prof. Dr. Jens-Peter Redlich
Frank Morgner

eingereicht am: ..... verteidigt am: .....

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

2FA Zwei-Faktor-Authentifizierung

AAGUID Authenticator Attestation Globally Unique Identifier

AES Advanced Encryption Standard

APDU Application Protocol Data Unit

API Application Programming Interface

ASM authentifikatorspezifisches Modul

AT Attested Credential Data

Bluetooth LE Bluetooth Low Energy

BSI Bundesministerium für Sicherheit in der Informationstechnik

CA Certificate authority

CBOR Concise Binary Object Representation

CC Common Criteria

COSE CBOR Object Signing and Encryption

CPU Central Processing Unit

CSRF Cross-Site Request Forgery

CTAP Client to Authenticator Protocol

EAL Evaluation Assurance Level

ECDAA Elliptic Curve based Direct Anonymous Attestation

ED Extension Data

EEPROM Electrically Erasable Programmable Read-Only Memory

FAR False Accept Rate

FBI Federal Bureau of Investigation

FIDO Fast Identity Online

FRR False Reject Rate

HID Human Interface Device

HMAC Hash-Based Message Authentication Code

HOTP HMAC-Based One-Time Password

ID Identifikator

ISO International Organisation for Standardization

ITU International Telecommunication Union

ITU-T ITU Telecommunication Standardization Sector

JCOP Java Card OpenPlatform

JCRE Java Card Runtime Environment

JSON JavaScript Object Notation

LSB Least Significant Bit

MAC Message Authentication Code

MCU Mikrocontroller

MFA Multi-Faktor-Authentifizierung

MSB Most Significant Bit

NFC Near Field Communication

NIST National Institute of Standards and Technology

OAuth Open Authorization

OTP One-Time Password

PC Personal Computer

PIN Personal Identification Number

PKI Public-Key-Infrastruktur

PSD2 Payment Service Directive 2

RAM Random-Access Memory

RFC Request for Comments

RK Resident Key

ROE restriktive Ausführungsumgebung

ROM Read-Only Memory

SAML Security Assertion Markup Language

SE sicheres Element

SG Security Goal

SIM Subscriber Identity Module

SSO Single Sign-On

TEE Trusted Execution Environment

TLS Transport Layer Security

TLV Tag-Length-Value

TOTP Time-Based One-Time Password

TPM Trusted Platform Module

U2F Universal 2nd Factor

UAF Universal Authentication Factor

UP User Present

URL Uniform Resource Locator

USB Universal Serial Bus

UV User Verified

UV-Strahlung Ultraviolettstrahlung

VM Virtual Machine

W3C World Wide Web Consortium

WebAuthn Web Authentication

XSS Cross-Site-Scripting

1 Einleitung

Die digitale Welt findet immer mehr Einzug in das Leben und Wirken der Menschen. Dabei fordern sowohl das soziale Umfeld als auch das Arbeitsumfeld immer mehr Teilhabe an der digitalen Welt. Soziale Netzwerke, Kalender, E-Mail-Anbieter, Spiele, Online-Shops, Banken und viele weitere Anwendungen erfordern dabei die Registrierung eines Accounts zur Nutzung des entsprechenden Dienstes. Während der Registrierung wird der Benutzer in der Regel dazu aufgefordert, einen Benutzernamen und ein dazugehöriges Passwort für den zu erstellenden Account festzulegen. Dabei hat jeder Dienst eigene Anforderungen, wie das Passwort zusammengesetzt sein muss. Beispielhafte Vorgaben wären eine Mindestlänge des Passworts von 8 Zeichen, welches jeweils mindestens einen Klein- und Großbuchstaben, eine Zahl und ein Sonderzeichen enthalten muss. Neben den Anforderungen werden von den Diensten sowie von anderen Einrichtungen wie dem Bundesministerium für Sicherheit in der Informationstechnik (BSI) [10] Anmerkungen herausgegeben, wie ein sicheres Passwort zusammengesetzt sein sollte. Zum Beispiel sollten keine Wörter, welche in einem Wörterbuch gelistet werden oder mit personenbezogenen Informationen verbunden sind, vorkommen, beispielsweise „DasPasswortIstSicher“ oder „*Eigener Vorname*“. Zudem sollte kein Passwort benutzt werden, das bereits für einen anderen Dienst verwendet wird und es sollte diesem auch nicht ähneln. Werden alle Vorgaben eingehalten, führt dies dazu, dass eine große Ansammlung an Benutzerdaten entsteht, welche sich der Anwender merken muss.

Diese große Anzahl an Passwörtern stellt für viele Benutzer eine große Herausforderung dar. Eine Studie von Florencio und Harley [11] aus dem Jahr 2006, bei der das Passwortnutzungsverhalten von über 540000 Teilnehmern ausgewertet wurde, zeigt, dass ein durchschnittlicher Internetbenutzer bei 25 Diensten registriert ist, jedoch im Mittel nur 6,5 verschiedene Passwörter besitzt. Jedes dieser Passwörter wird dabei für etwa 3,9 Dienste angewendet und hat eine Länge von im Schnitt 40,54 Bit, was ca. 5 Zeichen entspricht. Bestehen keine Anforderungen des Dienstes an das Passwort, werden zudem häufig nur Kleinbuchstaben durch den Benutzer gewählt. Werden hier die Passwörter der Längen von 6 bis 13 verwendeten Zeichen betrachtet, wird die minimale Anzahl ausschließlich aus Kleinbuchstaben bestehender Passwörter bei 8 Zeichen mit einem Anteil von 65% erreicht. Bei allen anderen Passwortlängen ist der Anteil ausschließlich aus Kleinbuchstaben bestehender Passwörter größer.

Aus den zuvor genannten Zahlen lässt sich daher schließen, dass eine Überforderung der Nutzer bzgl. der zu merkenden Passwörter besteht. Hinweise zur Passwortwahl werden hierbei häufig missachtet und nur die minimalen Anforderungen erfüllt. Beachtet man, dass die Daten 2006 erhoben wurden, ist die Anzahl der aktiven Webseiten im Internet seitdem auf das 4,45-fache gestiegen, wie aus dem September 2019 Web Server Survey von netcraft [12] hervorgeht. Waren es zu Beginn der Studie im Juli 2006 noch 42 258 721 aktive Webseiten, die registriert wurden, sind es 188 446 404 aktive Seiten im September 2019. Dieser deutliche Anstieg an aktiven Webseiten lässt ebenfalls auf eine Zunahme an Diensten, die eine Authentifizierung benötigen, schließen.

Um dieser Problematik entgegenzutreten, existieren bereits produktiv eingesetzte Verfahren, wie beispielsweise Passwortmanager [13, 14] und Single Sign-On [15]. Hierbei

soll durch die beiden genannten Verfahren die Anzahl der tatsächlich zu merkenden Passwörter reduziert werden – im besten Fall auf nur noch ein (Master-)Passwort. Dieses kann nun besonders stark gewählt werden und bietet damit entsprechende Sicherheit. Einen weiteren Ansatz stellt die Zwei-Faktor-Authentifizierung (2FA) dar, deren Verwendung mittlerweile auch in Richtlinien wie der Payment Service Directive 2 (PSD2) [16] der EU vorgeschrieben wird. Bei dieser Lösung wird während der Anmeldung ein weiterer Faktor in den Authentifizierungsprozess eingebracht, d.h. neben dem Passwort wird z.B. der Fingerabdruck des Benutzers zur Authentifizierung benötigt. Dies erhöht die Sicherheit des Accounts, da das Erraten bzw. Kompromittieren des Passworts durch einen Angreifer nicht mehr ausreicht, um Zugang zu dem geschützten Dienst zu bekommen. Zusätzlich müsste der zweite Faktor ebenfalls kompromittiert werden, um den Angriff erfolgreich zu beenden. Einer der Standards zur 2FA ist Universal 2nd Factor (U2F) [17], der von der Fast Identity Online (FIDO) Alliance entwickelt wird. Jedoch erfordern die genannten Verfahren – in der Regel – weiterhin die Nutzung eines Passworts.

Damit auch diese Passwörter nicht mehr notwendig sind, stellt die FIDO Alliance ebenfalls Standards zur passwortlosen und ggf. benutzerdatenlosen Authentifizierung gegenüber Diensten bereit. Diese sind der Standard Universal Authentication Faktor (UAF)[1] mit Fokus auf Smartphones sowie das Client to Authenticator Protocol (CTAP) [5] zusammen mit dem Standard Web Authentication (WebAuthn) [4]. CTAP und WebAuthn werden dabei unter dem Namen FIDO2 zusammengefasst. Seit März 2019 zählt WebAuthn zu den offiziellen Standards des World Wide Web Consortiums (W3C) [18]. Dieser Standard beschreibt eine Schnittstelle zur Integration des FIDO2-Protokolls CTAP in Online-Dienste und bietet zudem eine Beschreibung der entsprechenden Authentifikatoren (auch: Token, Sicherheitsschlüssel), um so die passwortlose bzw. benutzerdatenlose Anmeldung zu ermöglichen. Ziel ist es hierbei, eine einheitliche Infrastruktur für Webseitenbetreiber und Nutzer über viele Dienste hinweg zu ermöglichen. Neben der passwortlosen bzw.benutzerdatenlosen Anmeldung kann ein FIDO2-Token ebenfalls für die 2FA benutzt werden. Zudem regelt CTAP ebenfalls die Abwärtskompatibilität zu U2F, sodass alte 2FA-Tokens auch über die neue Schnittstelle als Zweit-Faktor verwendet werden können.

Im Rahmen dieser Arbeit wurde ein portables FIDO2-Token entwickelt, welches passwortlose Anmeldungen durch biometrische Authentifizierung mittels einer kontaktlosen Smartcard mit Fingerabdruck per NFC ermöglicht. Portable Tokens können dabei für alle Dienste angewendet werden, welche die Schnittstelle WebAuthn implementieren und eine entsprechende Authentifizierung mittels FIDO2 zulassen. Für die biometrische Authentifizierung wird auf der Smartcard ein Fingerabdrucksensor verbaut, sodass die Abnahme und Überprüfung des Fingerabdrucks ausschließlich auf der Smartcard erfolgt. Eine solche Smartcard wird im Weiteren auch als Fingerabdruckkarte bezeichnet. Die Arbeit zeigt damit, dass FIDO2 ohne interne Energiezufuhr, wie z.B. durch eine Batterie, und ohne physische Verbindung zum Clienten mit biometrischen Sensoren angewendet werden kann und so die passwortlose Authentifizierung ermöglicht. Zudem wurde eine Untersuchung der Zertifizierbarkeit des Tokens nach den Richtlinien der FIDO Alliance durchgeführt [6, 19, 20].

Im Folgenden wird auf die Entwicklung des Tokens und die verschiedenen Standards, sowie die Untersuchung der möglichen Zertifizierbarkeit des Tokens eingegangen. Dazu werden zu Beginn der Arbeit alternative Ansätze zur Reduzierung der Passwortmenge und anschließend verwandte Arbeiten vorgestellt. Darauf aufbauend werden die Forschungsbeiträge der Arbeit betrachtet. Nachfolgend werden der zu FIDO2 gehörige Standard WebAuthn sowie die CTAP-Protokolle der FIDO Alliance ausführlich dargestellt. Anschließend wird auf den durch die FIDO Alliance festgelegten Zertifizierungsprozess eines Tokens und die definierten Zertifizierungslevel eingegangen. Zum besseren Verständnis der Umsetzung wird daraufhin ein kurzer Überblick über die Grundlagen von Smartcards sowie über Biometrie gegeben. Nachdem alle notwendigen Grundlagen erläutert wurden, werden die Implementierung und die Designentscheidungen des FIDO2-Tokens dargestellt und begründet. Es folgt die Auswertung der Untersuchungen des Tokens bzgl. dessen Zertifizierbarkeit. Abschließend werden die Erkenntnisse der Arbeit zusammengefasst und ein Ausblick gegeben.

2 Hintergrund

Verschiedene Studien [11, 21, 22, 23] untersuchten bereits die Sicherheit und den Umgang mit von Nutzern gewählten Passwörtern zum Schutz ihrer Accounts. Hierbei fällt auf, dass alle Studien zu dem Schluss kommen, dass besonders das Gedächtnis des Menschen ein Problem darstellt. Aufgrund der stetig wachsenden Anzahl an Benutzeraccounts und der damit verbundenen Anzahl zu merkender Passwörter werden Strategien entwickelt, sodass die Erinnerung an Passwörter leichter fällt. Dazu zählen z.B. Methoden wie Wiederverwendung und Versionierung durch das Anhängen von Zeichenketten [22], aber auch eine grundsätzliche Wahl kurzer, schwacher Passwörter [11]. Zur Unterstützung des Gedächtnisses wird das Passwort zudem von 35,9% der befragten Benutzer auf einem Zettel notiert [23]. Aus diesen Vorgehensweisen resultieren erhebliche Sicherheitslücken für Accounts.

Den so entstehenden Sicherheitslücken stehen immer effektivere Angriffe gegenüber, wie die Studie von Han et al. [24] zeigt. Dazu zählen nicht nur manuelle Angriffe, wie das Suchen nach Notizzetteln, dem Beobachten bei der Passworteingabe sowie dem Erraten durch Social Engineering oder Phishing, sondern auch voll automatisierte Angriffsmethoden. Hierzu zählen z.B. Brute-Force-Angriffe, Wörterbuch-Attacken und Rainbow-Tables. Hierbei wird nach bestimmten Schemata versucht, das Passwort zu erraten, indem eine Liste von Passwörtern verwendet oder generiert wird, die online gegen den Dienst oder offline gegen einen erbeuteten Auszug der Datenbank des Servers ausprobiert werden können. Hier steigt die Effektivität der Angriffe mit der steigenden Rechenleistung durch Prozessoren und Grafikkarten weiter an. Es gilt daher Verfahren und Methoden zu finden, welche die Sicherheit von Accounts erhöhen und eine einfache Nutzbarkeit aufweisen.

Im Rahmen dieser Entwicklungen schlossen sich 2009 PayPal und Validity Sensors zusammen, um einen Lösungsansatz für passwortlose Authentifikation mittels Biometrie zu entwickeln, welche in der 2012 gegründeten FIDO Alliance resultierte [18].

Mittlerweile findet die FIDO Alliance Unterstützung in ihren Bestrebungen aus Wirtschaft und Regierungen, sowie anderen Standardisierungsgremien. Zu den mittlerweile über 250 Mitgliedern gehören u.a. Google, Samsung, das Bundesamt für Sicherheit in der Informationstechnik [25] sowie kooperierende Gremien wie das World Wide Web Consortium (W3C) [26]. Aufgrund dieser breiten Unterstützung sind die von der FIDO Alliance entwickelten Standards von internationalem Interesse, um ein einheitliches Ökosystem für sichere Authentifizierung zu schaffen. Zuletzt wurde im März 2019 der FIDO2-Standard, bestehend aus Web Authentication (WebAuthn) [4] und dem Client to Authenticator Protocol (CTAP) [5], offiziell durch das W3C und die FIDO Alliance verabschiedet. Der Standard umfasst sowohl ein Protokoll zur passwort- oder benutzerdatenlosen Authentifizierung mittels Biometrie als auch ein Protokoll, welches die Abwärtskompatibilität zu U2F-Tokens bereitstellt.

Neben dem in dieser Arbeit vorgestellten Ansatz, welcher auf den Standards der FIDO Alliance basiert, existieren weitere Versuche, die Anmeldung mittels Benutzernamen und Passwort sicher und einfach zu gestalten. Dabei wird versucht, die benötigte Anzahl an Passwörtern zu reduzieren bzw. die Verwendung von Passwörtern zu eliminieren. Gleichzeitig soll die Sicherheit des Benutzeraccounts durch Verwendung der vorgestellten Verfahren steigen. In diesem Abschnitt werden einige dieser Ansätze, wie z. B. Multi-Faktor-Authentifizierung, Passworttresore und Single-Sign-On, vorgestellt. Abschließend werden die vorherigen Standards der FIDO Alliance vorgestellt sowie auf deren Verbreitung und auf ähnliche Produkte eingegangen.

2.1 Alternative Lösungsansätze

Der Zugang zu einem Dienst muss nicht ausschließlich über das Setzen eines Passworts abgesichert werden. Betrachtet man z. B. Smartphones, stellen viele Hersteller mittlerweile neben der klassischen PIN weitere Verfahren zum Entsperren des Gerätes bereit. Beispielsweise kann hier der Zugang per Fingerabdruck oder Gesichtsbiometrie gewährt werden, welche der Nutzer zuvor auf dem Smartphone hinterlegt. Während die PIN zu den Geheimnissen der Kategorie „Wissen “ zählt, kann der Fingerabdruck nicht „gewusst werden“. Der Fingerabdruck selbst stellt eine einzigartige, biometrische Eigenschaft des Besitzers dar, welche ihn eindeutig identifiziert [27, Kapitel 10]. Somit gibt es neben der Kategorie „Wissen“ weitere Gruppen, aus denen Authentifikatoren gewählt werden können. Nach Ometov et al. [28] wird in folgende Gruppen unterschieden:

Wissen: alleiniges Wissen des rechtmäßigen Benutzers (z. B. ein Passwort)

Besitz: alleiniger Besitz des rechtmäßigen Benutzers (z. B. ein kryptographisches Hardware-Token)

Biometrie: biometrisches Merkmal des rechtmäßigen Benutzers (z. B. ein Fingerabdruck)

Nachfolgend werden die Multi-Faktor-Authentifizierung sowie einige weitere Verfahren vorgestellt.

2.1.1 Multi-Faktor-Authentifizierung

Die Multi-Faktor-Authentifizierung (MFA) [28, 29] wird angewendet, um die Sicherheit bei einem Anmeldevorgang durch die Nutzung mehrerer Faktoren zu steigern. Das heißt, es wird bei einem Anmeldeversuch durch einen Benutzer, beispielsweise neben dem standardmäßig hinterlegten Passwort, mindestens ein weiterer Authentifizierungs-Faktor abgefragt. Dabei wurde dieser zusätzliche Faktor zuvor von dem rechtmäßigen Benutzer des Accounts bei dem Service hinterlegt. Bei der Wahl sollte darauf geachtet werden, dass Faktoren verschiedener Gruppen, wie unter 2.1 definiert, kombiniert werden, um eine erhöhte Sicherheit zu gewährleisten. Dies erschwert einem Angreifer einen erfolgreichen Identitätsdiebstahl. Zudem hängt die Sicherheit des Dienstes nicht nur von der Anzahl und der Gruppe der gewählten Faktoren, sondern auch von den verwendeten Faktoren selbst ab. Oft findet die Multi-Faktor-Authentifizierung in Form der Zwei-Faktor-Authentifizierung (2FA) Anwendung, bei der genau zwei Faktoren für die Anmeldung festgelegt werden.

Damit eine Registrierung zusätzlicher Faktoren möglich ist, muss sich der Benutzer zuerst mittels des standardmäßig gewählten Anmeldeverfahrens gegenüber dem Dienst authentifizieren. War die Authentifizierung erfolgreich, können ein oder mehrere zusätzliche Faktoren für den Account hinterlegt werden. Ist dies geschehen, werden bei einem erneuten Anmeldeversuch der neu hinterlegte Faktor bzw. die neu hinterlegten Faktoren ebenfalls abgefragt. Nur wenn alle Faktoren zum Zeitpunkt der Anmeldung vorhanden sind, wird diese erfolgreich durchgeführt [28].

Auf der einen Seite ermöglicht die Multi-Faktor-Authentifizierung die Verwendung schwacher Passwörter, da ein Erraten des Passworts nicht mehr ausreicht, um sich dem Dienst gegenüber zu authentifizieren. Wird also das Passwort durch einen Angreifer erraten, reicht dies nicht mehr aus, um den Identitätsdiebstahl zu begehen. Somit muss sich der Benutzer theoretisch nur noch leichte Passwörter merken. Jedoch sollten die für die MFA verwendeten Faktoren selbst entsprechend stark sein, um hier ein hohe Sicherheit zu erreichen, d. h. das verwendete Passwort sollte entsprechend stark gewählt werden [17] [30, Kapitel 5]. Auch das theoretische Ersetzen des Passworts durch einen anderen Faktor ist mittels MFA möglich. Somit wird bei der Registrierung kein Passwort mehr abgefragt und die potentielle Stärke eines Passworts ist nicht mehr relevant für die Sicherheit des Accounts. Der Nutzer muss sich also keine Passwörter mehr überlegen und merken. Jedoch findet dies heutzutage kaum Anwendung. In den meisten Fällen verlangen die Dienste weiterhin die übliche Kombination aus Benutzernamen und Passwort als ersten Faktor und lassen nur für den zweiten Faktor eine begrenzte Wahl durch den Benutzer zu [28]. Weiterhin erfordert der Authentifizierungsprozess verschiedene Nutzerinteraktionen, bei denen die verschiedenen Faktoren Anwendung finden. Versucht ein Angreifer den Zugang zu einem Account zu erhalten, ohne alle Faktoren zu besitzen, wird somit ggf. der rechtmäßige Besitzer – je nach gewählten Faktoren – über einen unrechtmäßigen Anmeldeversuch frühzeitig informiert [30, Kapitel 5] [31]. Somit profitiert durch die Anwendung von MFA sowohl die Sicherheit des Accounts als auch der Benutzer, der sich an weniger oder leichtere Zugangsdaten erinnern muss.

Andererseits bestehen bei Verwendung der MFA ebenfalls Probleme hinsichtlich der Nutzbarkeit, Robustheit und Sicherheit [28]. Je mehr Faktoren der Benutzer für den Anmeldevorgang hinterlegt, desto umständlicher und langwieriger wird der Anmeldeprozess, da alle Faktoren zum Zeitpunkt des Logins vorhanden sein und einzeln gegen den Server geprüft werden müssen. Erst wenn dies erfolgreich durchgeführt wurde, wird die Anmeldung ausgeführt. Es müssen also jederzeit die entsprechenden Faktoren verfügbar sein, falls eine Anmeldung notwendig wird [32, 33]. Zudem kann es einem Angreifer gelingen, in den Besitz aller Faktoren zu gelangen und somit unbemerkt einen Angriff auszuführen. Dies ist z.B. der Fall, wenn nur unzureichende Faktoren oder Faktoren einer einzigen Kategorie verwendet werden. Dies wiegt den Nutzer in falscher Gewissheit bzgl. des Schutzes seines Accounts [31, 34]. Um bei der Authentifizierung weitere Probleme zu vermeiden, sollte zusätzlich ein alternatives Verfahren oder ein alternativer Faktor für den Anmeldevorgang hinterlegt werden. Wird ein Faktor vergessen, verloren oder ist dauerhaft nicht mehr nutzbar, so kann das alternative Verfahren für die Authentifizierung gegenüber dem Dienst genutzt werden. Dies erfordert jedoch das zusätzliche Vorhalten des entsprechenden Faktors. Wird für diesen Schritt jedoch bspw. ein (schwaches) Passwort gewählt, gehen alle Vorteile des vorherigen Verfahrens verloren. Ein Angreifer könnte in diesem Fall so lange die Anmeldung versuchen, bis auf das alternative Verfahren zurückgegriffen wird und so einen gegebenenfalls leichteren Angriff durchführen. Existiert keine entsprechende Alternative, ist der Zugang selbst für den rechtmäßigen Benutzer nicht mehr möglich und muss erneut eingerichtet werden. Auch die Robustheit der einzelnen Faktoren kann ein Problem darstellen – wird beispielsweise der Fingerabdruck als biometrischer Faktor verwendet, sollten die Sensoren entsprechend robust sein und den Fingerabdruck auch dann verifizieren können, wenn er anders auf den Sensor gelegt wird als bei der Einrichtung des Faktors. Hierbei muss zudem ausgeschlossen werden, dass ein falscher Finger als rechtmäßig erkannt wird [28].

2.1.2 Einmalpasswort

Das Einmalpasswort, auch One-Time Password (OTP) [35] genannt, ist ein für den aktuellen Anmeldevorgang einmalig gültiges Passwort. Möchte der Benutzer eines Dienstes eine Anmeldung durchführen, wird durch das System ein nur für diesen Vorgang gültiges Kennwort generiert und dem Nutzer übermittelt. Dieses Passwort muss anschließend für die Authentifizierung gegenüber dem Dienst verwendet werden. Wurde die Authentifizierung erfolgreich durchgeführt und/oder ist eine frei definierte Zeit verstrichen, ist das Passwort ungültig und es muss für den nächsten Versuch ein neues Passwort angefragt werden. Dies verspricht eine höhere Sicherheit, als die Verwendung von herkömmlichen wiederverwendbaren Passwörtern, da hier das Passwort durch den Angreifer nicht erraten und anschließend dauerhaft für die Anmeldung benutzt werden kann. Einmalpasswörter können zudem als ein Faktor der Multi-Faktor-Authentifizierung genutzt werden.

Zur Generierung von Einmalpasswörtern stehen verschiedene Verfahren zur Verfügung. Wie in [35] durch Rubin beschrieben, existieren sowohl software- als auch

hardwarebasierte Ansätze. Softwarebasierte Ansätze werden beispielsweise durch einige Standards wie das RFC 1760 [36] und das RFC 2289 [37], auch S/KEY bzw. OTP genannt, aufgegriffen. Die dort beschriebenen Systeme generieren voneinander abhängige Einmalpasswörter und beruhen darauf, dass auf einem Server durch den Benutzer einmalig OTPs generiert werden, welche anschließend verwendet werden können. Dazu wählt der Benutzer auf dem Server ein Passwort als Grundlage der Berechnung sowie eine Anzahl n an zu generierenden Einmalpasswörtern. Anschließend wird das angegebene Passwort n-mal mit einer Einweg-Hashfunktion gehasht und das Ergebnis sowie die Anzahl n auf dem Server hinterlegt. Möchte sich der Benutzer nun authentifizieren, gibt dieser das zur Generierung der Einmalpasswörter verwendete Passwort ein und der Client hasht dieses $n-1$ mal. Das so generierte OTP wird anschließend an den Server gesendet. Der Server hasht daraufhin das vorliegende OTP das n-te Mal und vergleicht den gespeicherten Hash mit dem nun vorliegenden Hash. Ist dieser gleich, ist die Authentifizierung erfolgreich und anschließend wird der erhaltene $n-1$-te Hash auf dem Server hinterlegt und der Wert $n-1$ gespeichert.

Einen anderen Ansatz stellen z.B. HMAC-basierte (HOTP) und zeitbasierte (TOTP) OTPs dar, welche in RFC 4226 [38] und RFC 6238 [39] beschrieben werden. Diese Algorithmen sind dabei sowohl für die Verwendung in Hardware als auch in Software geeignet, wie z.B. als Smartphone-App [40]. Beide Verfahren basieren darauf, dass sowohl auf dem Authentifizierungsserver als auch auf dem entsprechenden Hardware-Token bzw. in der entsprechenden Smartphone-App ein zuvor geteiltes, symmetrisches Geheimnis hinterlegt wird. Anschließend werden Server und Token mittels Zähler (HOTP) oder zeitbasiert (TOTP) synchronisiert, sodass Token und Server zu jeder Zeit das gleiche Geheimnis ableiten. D.h. bei HOTP wird bei jedem Anmeldeversuch ein entsprechender Zähler hochgezählt, welcher in die Berechnung des neuen Einmalpasswortes, ausgehend von dem geteilten Geheimnis, einbezogen wird. Bei TOTP wird hingegen nach einem frei definierten Zeitintervall, z.B. alle 30 Sekunden, ein neues Einmalpasswort generiert – hier spielt jedoch die Synchronisation der Uhren eine wichtige Rolle.

Im Gegensatz zu herkömmlichen Passwörtern bieten Einmalpasswörter einige Vorteile [35]. Dadurch, dass sie nur genau einmalig gültig sind, stellt ein nachträglicher Verlust oder ein nachträgliches Bekanntwerden der Passwörter kein Problem dar. Durch die ständige Änderung der Passwörter werden zudem Meet-in-the-middle-Angriffe erschwert. Des Weiteren spielt der Faktor Mensch bei der Wahl der Passwörter keine Rolle mehr. Dies führt zu einer höheren Sicherheit der Systeme, da eine Wahl von schwachen, wiederverwendbaren Passwörtern nicht möglich ist. Wörterbuchangriffe, Passwortlisten oder ähnliche Angriffsszenarien sind somit auf Dauer unpraktikabel.

Nachteile, wie in [35] diskutiert, ergeben sich jedoch daraus, dass einige Verfahren, wie die oben erwähnten, auf von Benutzer gewählten Geheimnissen basieren, die jedes Mal zur Generierung des aktuellen Einmalpasswortes verwendet werden. Diese verhalten sich somit wie herkömmliche Passwörter, die es zu schützen gilt. Da zur Generierung ebenfalls das Hash-Verfahren und die aktuell durchzuführende Anzahl der Hash-Durchläufe bekannt sein müssen oder der verwendete Client des Benutzers verfügbar sein muss, reicht alleine das Passwort nicht aus, um einen Angriff erfolgreich durchzuführen. Gleiches gilt bei Bekanntwerden von Seeds, die in anderen Verfahren

wie HOTP und TOTP festgelegt werden. Sind alle Parameter des Dienstes bekannt, wie z.B. verwendete Hash-Funktionen, Zeitintervalle etc., können Angreifer entsprechende Einmalpasswörter selbst generieren. Sollen zudem mehrere Authentifizierungsserver von Seiten des Dienstes bereitgestellt werden, so muss eine korrekte Synchronisierung der Authentifizierungsserver gewährleistet werden. Ist dies nicht der Fall, können Replay-Angriffe oder Probleme bei der Authentifizierung durch den legitimen Benutzer nicht ausgeschlossen werden. Wie in [40] beschrieben, spielt auch die Sicherheit der Plattform, auf der das Einmalpasswort generiert wird, eine wichtige Rolle. Ist die Plattform kompromittiert oder wird ein Denial-of-Service-Angriff auf den Dienst ausgeführt, kann dies zu Fehlern in der Generierung der Einmalpasswörter bis hin zum Verlust von Seeds oder Einmalpasswörtern führen. Die Vertraulichkeit der Einmalpasswörter ist somit nicht mehr gewährleistet.

2.1.3 Passwortmanager

Die Grundfunktion eines Passwortmanagers [13, 14] ist die sichere Speicherung von Benutzerdaten zu einem Dienst. Passwortmanager verwalten somit eine Datenbank mit Zugangsdaten des Benutzers, welche verschlüsselt gespeichert wird. Dazu wählt der Benutzer als Masterschlüssel ein möglichst sicheres Passwort. Dieses ist anschließend das einzige Passwort, dass sich der Benutzer fortan merken muss. Möchte er sich nun einem Dienst gegenüber authentifizieren, kann die Datenbank des Passwortmanagers mittels des Masterpassworts entschlüsselt werden. Ist dies erfolgt, werden die benötigten Zugangsdaten gesucht und das Passwort aus der Datenbank in das Anmeldeformular kopiert. Neben der reinen Speicherung von Passwörtern bieten Passwortmanager häufig die Generierung starker Passwörter an, welche während der Registrierung oder nachträglich bei einem Dienst hinterlegt werden können. Somit ist es dem Benutzer möglich, für jeden Dienst ein einzigartiges, starkes Passwort zu generieren, welches es einem Angreifer erschwert, einen erfolgreichen Angriff auszuführen. Passwortmanager sind unter anderem sowohl für Smartphones, als Desktopanwendung, als Integration in Betriebssysteme, wie z.B. Apple, oder Browser, wie bspw. Firefox, oder als plattformübergreifende Anwendungen mit zentraler Datenbank verfügbar. Je nach gewählter Anwendung ermöglicht eine Synchronisation zwischen den einzelnen Clienten somit den komfortablen Zugriff auf die Datenbank. Benutzerdaten müssen daher nur einmalig eingepflegt werden und können anschließend überall, wo Zugriff auf die Datenbank besteht, abgerufen werden.

Ein Passwortmanager bringt demzufolge viele Vorteile mit sich [13, 14]. Die Datenbank ermöglicht es, sichere und starke Passwörter für die einzelnen Dienste zu generieren, ohne dass diese dem Benutzer bekannt sein müssen. Dies führt dazu, dass Wörterbuch- und Brute-Force-Angriffe auf die mit dem Passwortmanager verwalteten Dienste wesentlich schwerer werden. Sollte dennoch ein Angriff erfolgreich sein, ist nur ein Dienst betroffen und nicht mehrere Dienste, bei denen das gleiche Passwort verwendet wird. Einzig das Masterpasswort ist für den Benutzer relevant und sollte, da sich der Benutzer keine weiteren Passwörter mehr merken muss, entsprechend komplex gewählt

werden. Einmal eingerichtet kann der Passwortmanager zudem – je nach Anwendung und Plattformunterstützung – auf verschiedenen Geräten verwendet werden.

Aus der Studie von Alkaldi und Renaud [14] geht hervor, dass nur wenige Nutzer Passwortmanager verwenden. Von 836 befragten Personen hat nur ca. 1% der Befragten angegeben, einen Passwortmanager zu nutzen. Hier werden negative Erfahrungen, fehlendes Bewusstsein, Sicherheitsbedenken, fehlendes Vertrauen in die Software, ein geringer Schutzbedarf sowie anfallende Kosten und die Benutzbarkeit als die häufigsten Argumente angegeben. Ein mögliches Bedenken ist z. B. die Offenbarung aller Passwörter, sollte es einem Angreifer gelingen, das Masterpasswort der Datenbank zu erhalten. Dies deckt sich mit den Ergebnissen der Studie von Zhang et al. [13]. Neben den vorherigen Erkenntnissen befasst sich die Studie zudem mit der Art und Weise, wie Benutzer Passwortmanager verwenden. Die Studie ergab, dass trotz der Verwendung von Passwortmanagern die Wiederverwendung von Passwörtern stattfindet. Bereits bestehende Passwörter wurden in den meisten Fällen nicht geändert und nur neu registrierte Dienste erhielten neue, zufällig generierte Passwörter. Bei der Vergabe des Massterpassworts wählte die Mehrheit der Befragten jedoch ein zuvor noch nicht verwendetes Passwort.

Neben den zuvor erörterten Problemen mit der Ablehnung von Passwortmanagern und dem falschen Umgang durch die Benutzer, existieren weitere Sicherheitsrisiken. So konnte von Silver et al. [41] gezeigt werden, dass die Auto-Fill-Funktion, welche moderne Passwortmanager anbieten, auf strukturellen Eigenschaften sowie der URL des Dienstes aufbaut. Somit ist es einem Angreifer möglich, beispielsweise Passwörter mittels Cross-Site-Scripting (XSS) über die Auto-Fill-Funktion zu extrahieren. Eine weitere Studie von Gray et al. [42] kam zu dem Ergebnis, dass bei einigen in der Studie untersuchten Passwortmanagern Daten im Klartext aus dem Dateisystem des Anwenders ausgelesen werden konnten – selbst wenn der Passwortmanager bereits beendet wurde. Dazu zählten temporäre Dateien oder Dateien im Papierkorb, welche die Daten im Klartext enthielten. Abschließend sei hier die Studie von Gasti und Rasmussen [43] erwähnt. Gasti und Rasmussen analysierten die Datenbankformate unterschiedlicher Passwortmanager und konnten so zeigen, dass die meisten Datenbankformate unzureichend vor Angriffen geschützt sind. Beispielsweise wird bei der Verwendung des in Firefox integrierten Passwortmanagers die Integrität der zu den Benutzerdaten gehörigen URL nicht geschützt. Ein Angreifer mit Zugriff zum System des Benutzers kann folglich diesen Eintrag ändern und einen Phishing-Angriff versuchen.

Trotz der möglichen Sicherheitsrisiken wird die Nutzung von Passwortmanagern durch Experten weiterhin empfohlen [13].

2.1.4 Single Sign-On

Single Sign-On (SSO) [15] beschreibt den Umstand einer gemeinsamen Authentifizierungsquelle für mehrere Dienste, bei der die einmalige Authentifizierung des Benutzers gegenüber der Quelle ausreicht, um alle mit der Quelle verbundenen Dienste nutzen zu können. Ein Benutzer kann somit seine Identität mit nur einem einzigen Authentifizierungsvorgang gegenüber verschiedenen Diensten bestätigen und diese damit ohne

weitere manuelle Authentifizierungsvorgänge nutzen. Soziale Netzwerke und große Firmen, wie beispielsweise Facebook und Google bieten bereits eigene Single-Sign-On-Services an, welche durch verschiedenste Dienste über eine bereitgestellte Schnittstelle integriert werden können [44]. Es reicht somit nur ein Facebook- oder Google-Account aus, um die mit dem Single-Sign-On-Service der jeweiligen Firma verbundenen Dienste zu benutzen. Dieses Verfahren bietet einen großen Komfort für Nutzer und findet für Webdienste große Beliebtheit – 77% der Nutzer bevorzugen mittlerweile die Verwendung von SSO für die Authentifizierung. Einige Protokolle für SSO-Lösungen sind z.B. die zwei auf OAuth 2.0 [45] basierenden Protokolle OpenID Connect [46] und SAML 2.0 [47], sowie Kerberos [48].

Die Grundlage für Single Sign-On Systeme bildet der Austausch von Tokens (auch Tickets) zwischen dem Clienten und einer oder mehreren zuständigen Autorisierungsstelle bzw. -stellen, wie z.B. für OAuth 2.0 [45] beschrieben. Diesem Beispiel folgend, wird in einem ersten Schritt das Recht zur Autorisierung gegenüber des Dienstes in Form eines Autorisierungstokens angefragt. Wurde das Recht in Form des Tokens bestätigt, kann der Client nun den Autorisierungsserver anfragen, welcher die in dem Autorisierungstoken enthaltenen Credentials auf Validität überprüft und bei Erfolg ein Zugangstoken zum Dienst bereitstellt. Nun kann der Client den Dienst nach bestimmten Ressourcen anfragen, indem das Zugangstoken mit der Anfrage zusammen an den Server des Dienstes übermittelt wird. Ist die Validierung des Zugangstokens erfolgreich, werden die angeforderten Ressourcen an den Clienten ausgeliefert. Der Benutzer kann den Dienst nun nutzen.

Neben den oben genannten Vorteilen, dass der Benutzer bestenfalls nur noch einen Account für die Verwendung verschiedener Dienste benötigt, dem Benutzer also nur noch ein Passwort bekannt sein muss, und der bereits großen Popularität und Benutzerfreundlichkeit der Lösung, lassen sich auch hier Sicherheitsrisiken feststellen. Wang et al. [44] sowie Sun und Beznosov [49] konnten dabei Schwachstellen in den Implementierungen der einzelnen Dienste feststellen. Dazu gehörten Sicherheitslücken durch fehlende Verschlüsselung des Netzwerkverkehrs beim Aushandeln von Tokens oder die fehlende Bindung der Tokens an kontextuale Daten des Nutzers und des Servers. Auch Angriffe mittels Cross-Site-Scripting (XSS), Imitation oder Force-Login Cross-Site Request Forgery (CSRF) sowie das Profiling des Nutzers sind möglich [49]. Bei XSS-Angriffen kann beispielsweise durch die Verwendung von iFrames auf böswilligen Seiten das Token ausgelesen werden. Die Imitation basiert auf dem Ansatz, dass SSO-Credentials zum Erhalt von validen Tokens erraten werden können – bei 9% der Dienste war dies z.B. mittels der öffentlich einsehbaren Account-ID des anzugreifenden Benutzers von Facebook möglich. Bei Force-Login CSRF hingegen, handelt es sich häufig um einen Angriff, bei dem ein bereits authentifizierter Nutzer unbemerkt eine böswillige Seite aufruft. Im Hintergrund werden nun gegen den SSO-Server des vom Nutzer verwendeten SSO-Dienstleisters unbemerkt Anfragen durch die böswillige Seite gestellt, um ein valides Autorisierungs- oder Zugangstoken zu erhalten. Insgesamt konnten im Rahmen der Studie bei 91% der untersuchten Dienste entsprechende Tokens extrahiert werden. Weiterhin besteht bei Benutzung des SSOs die Gefahr des Profilings eines Benutzers

durch den Dienstleister oder einen Angreifer, da dieser Informationen über die vom Benutzer verwendeten Dienste und dessen Aktivitäten sammeln kann.

2.2 Verwandte Arbeiten

Im Folgenden werden die Standards Universal 2nd Factor (U2F) [17] in der Version 1.2 vom 11.04.2017 und Universal Authentication Factor (UAF) [1] in der Version 1.1 vom 02.02.2017 betrachtet. Die Standards wurden erstmalig im Dezember 2014 durch die FIDO Alliance veröffentlicht [18]. U2F und UAF sind dabei die Vorgänger des Standards WebAuthn und der dazugehörigen CTAP-Protokolle. U2F standardisiert ein Verfahren zur Zwei-Faktor-Authentifizierung (2FA), während UAF eine passwortlose oder Multi-Faktor-Authentifizierung beschreibt. Dabei umfassen die Standards sowohl die Protokolle und Sicherheitsanalysen als auch die Schnittstellen für Dienste und Tokens. Es soll so ein Ökosystem geschaffen werden, welches eine sichere Authentifizierung überall im Internet ermöglicht, indem eine standardisierte Schnittstelle für Online-Dienste geschaffen wird. Ein kompatibles Token kann somit für jeden Online-Dienst verwendet werden, der die entsprechende Schnittstelle des Tokens bereitstellt. Um mit jeder Art von Endgerät zu funktionieren, wird der Transport über USB, Bluetooth oder NFC bereitgestellt [17, 50]. Tokens können z. B. in Form eines USB-Sticks, Smartphones, einer Smartcard oder eines Software-Tokens auftreten. Einzellösungen für verschiedene Dienste und Anbieter werden bei Verwendung von U2F und UAF unnötig. Zudem können sowohl U2F als auch UAF zusammen mit anderen Verfahren, wie die in Kapitel 2.1.4 genannten, kombiniert werden.

In Abbildung 1 wird der Ablauf eines Registrierungs- bzw. Authentifizierungsprozesses für U2F [2] und UAF [1] dargestellt. Zu Beginn (1) wird die Authentifizierung gestartet und eine Anfrage durch den Clienten, z. B. den Browser des Benutzers, an den FIDO-Server des Dienstes gesendet. Dieser antwortet mit einer Aufgabe (2). Anschließend wird die Aufgabe durch den Clienten an das Token weitergeleitet (3), welches die entsprechenden Schritte für die Authentifizierung anhand der Nachricht abarbeitet. Das Ergebnis wird zurück an den Clienten geschickt (4). Im nächsten Schritt sendet der Client die erhaltene Nachricht, ggf. mit weiteren vom FIDO-Server benötigten Informationen, an den Server zurück (5). Dieser verifiziert nun die Aufgabe, sowie die restlichen Daten (6). Ist die Verifikation erfolgreich, ist die Authentifizierung abgeschlossen. Zur Verifikation kann u.a. der FIDO-Metadaten-Service verwendet werden, welcher Metadaten über alle von der FIDO Alliance zertifizierten Tokens bereithält [51]. Der Metadaten-Service kann dabei sowohl von dem Dienstanbieter selbst als auch von einem vertrauenswürdigen Drittanbieter gestellt werden. Die Metadaten zu einem Token umfassen dabei den öffentlichen Schlüssel zu einem auf dem Token hinterlegten Zertifikat des Herstellers. Das Zertifikat ist hierbei eindeutig für die Geräteserie des Herstellers und wird über ein für die Geräteserie spezifisches Schlüsselpaar erstellt. Neben dem Zertifikat wird auch der private Schlüssel des Schlüsselpaares auf dem Token hinterlegt. Dadurch kann anschließend über die Zertifikatskette garantiert werden, dass das Token von einem rechtmäßigen Hersteller stammt. Es wird somit auf eine Public-

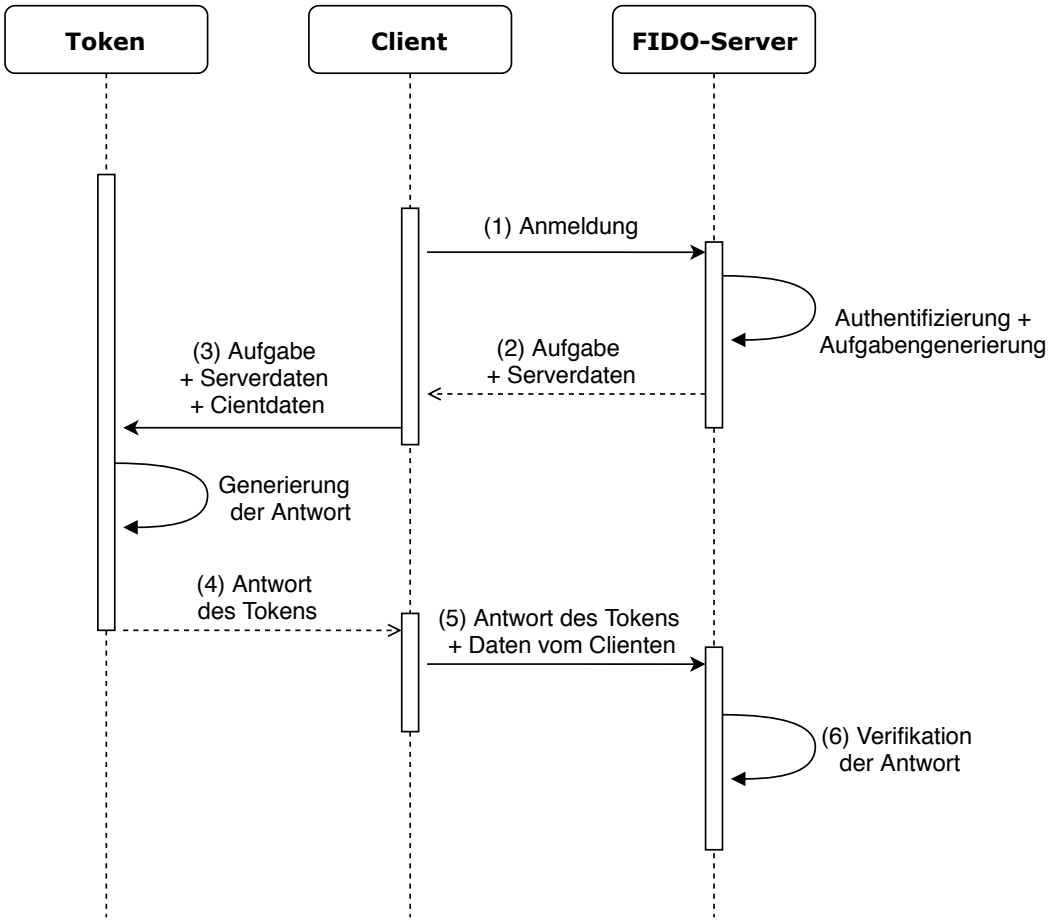

Abbildung 1: Allgemeines Sequenzdiagramm zum Ablauf der Protokolle U2F und UAF nach [1, 2]. Schritt (1) und (2) stellen die Setup-Phase, Schritt (3) und (4) die Processing-Phase des Tokens und Schritt (5) und (6) die Verifikationsphase dar.

Key-Infrastruktur (PKI) zurückgegriffen, welche die Vertrauenswürdigkeit eines Tokens bestimmt.

Der im vorherigen Absatz erklärte Ablauf wird in der Registrierungsphase zur Etablierung eines Schlüsselpaares verwendet. Dabei wird für jeden Dienst ein individuelles Schlüsselpaar generiert, damit ein Rückschluss auf den einzelnen Benutzer anhand gleicher Schlüsselpaare nicht möglich ist und somit die Privatsphäre gewährleistet wird. Der öffentliche Schlüssel wird, zusammen mit einem Key Handle, welches das Schlüsselpaar für das Token eindeutig identifiziert, immer auf Seiten des FIDO-Servers gespeichert. Dabei kann der private Schlüssel von dem Token verwaltet oder in dem Key Handle, durch ein vom Hersteller gewähltes Verfahren, verschlüsselt auf dem FIDO-Server hinterlegt werden. Das Schlüsselpaar wird vom Token zwischen Schritt (3) und (4) des in Abbildung 1 dargestellten Ablaufs generiert und dabei an das Token und den entsprechenden Online-Dienst gebunden. Für die Bindung des Schlüsselpaares an beide Parteien werden Hashes über Metadaten des FIDO-Servers sowie des Cli-

enten in die Generierung mit einbezogen. So entsteht ein einzigartiges Schlüsselpaar zwischen Token und Dienst. Wird ein Token für mehrere Accounts desselben Dienstes verwendet, entsteht für jeden Account ebenfalls ein eigenes Schlüsselpaar. Während der Authentifizierung wird anstelle der Etablierung eines Schlüsselpaares über mit dem privaten Schlüssel des Schlüsselpaares erstellte Signaturen nachgewiesen, dass das gerade verwendete Token bereits auf dem FIDO-Server registriert wurde. Dazu sendet der FIDO-Server entsprechend eine Aufgabe und den Server beschreibende Metadaten über den Clienten an das Token. Das Token generiert nun eine Signatur mittels des privaten Schlüssels über den Daten des FIDO-Servers und den zum Token gehörigen Metadaten. Anschließend werden die Metadaten und die Signatur an den FIDO-Server übermittelt, der nun die Daten verifiziert.

Trotz eines ähnlichen Protokollablaufs wird in keinem der Standards die Kompatibilität untereinander beschrieben. Somit müssen beide Schnittstellen unabhängig voneinander in Dienste integriert werden. Im Folgenden werden daher die Protokolle U2F und UAF genauer betrachtet, und die Besonderheiten der Protokolle getrennt dargestellt. Daraufhin wird kurz auf die Sicherheit der beiden Protokolle eingegangen. Abschließend wird die Verbreitung aller von der FIDO Alliance bereitgestellten Standards betrachtet und die von AuthenTrend entwickelte ATKey.card [52] vorgestellt.

2.2.1 Universal 2nd Factor

Der Standard U2F [17] beschreibt ein Verfahren der 2FA und lässt sich somit zu der in Kapitel 2.1.1 vorgestellten Multi-Faktor-Authentifizierung einordnen. U2F soll einem Dienst die Möglichkeit bieten, einen sicheren Login durch die Einbringung eines zusätzlichen Faktors während des Authentifizierungsprozesses zu gewährleisten. Nach erfolgter Authentifikation durch Benutzername und Passwort müssen die Tokens somit nur die Anwesenheit des Benutzers bestätigen können, ohne den Nutzer selbst zu authentifizieren. D. h. das simple Drücken eines Knopfes, das Berühren einer Schaltfläche oder das Einstecken eines USB-Sticks, während des Authentifizierungsprozesses, sind ausreichend. Somit können sich theoretisch auch mehrere Personen einen Faktor teilen, da der Benutzer nicht an das Token gebunden wird. Da bei jedem Registrierungsvorgang bei einem Dienst ein eigenes Schlüsselpaar etabliert wird, stellt dies auch in der Praxis kein Problem dar. Zur Verwendung wird, neben einem U2F-Token und einem Clienten, der U2F unterstützt, keine weitere Software, wie z.B. Treiber, benötigt. Die Schnittstelle soll im Clienten selbst integriert werden und über das Betriebssystem mit dem Token kommunizieren können.

Sind mehrere U2F-kompatible Faktoren an einem Gerät angeschlossen, so kann der Benutzer durch die Bestätigung der Anwesenheit an einem Token auswählen, dass dieser Faktor für die Authentifizierung genutzt werden soll. Hierbei muss darauf geachtet werden, dass, bei einer Anmeldung bei einem Dienst, ein bereits bei dem Dienst hinterlegtes Token gewählt wird. Andernfalls wird die Authentifizierung fehlschlagen. Seit Version 1.2 besteht zudem die Möglichkeit, dass Token auch ohne Bestätigung der Anwesenheit verwendet werden können, dies muss aus der durch das Token generierten Signatur jedoch eindeutig hervorgehen.

Neben dem Sicherheitsaspekt beschäftigt sich U2F auch mit der Privatsphäre des Nutzers. Das Protokoll ist so ausgelegt, dass ein Token keine globale ID besitzt, die bei Diensten hinterlegt werden kann. Somit kann bspw. ein Dienst, bei dem mehrere Accounts durch dasselbe Token abgesichert werden, keine Rückschlüsse darauf ziehen, dass es sich wirklich um dasselbe Token handelt. Weiterhin kann ein Dienst keine Informationen darüber erhalten, bei welchen weiteren Diensten das Token verwendet wird. Wird eine Authentifizierungsanfrage von einem Dienst gestellt, gleicht das Token den vom Clienten übermittelten Ursprung mit dem im Key Handle des Dienstes enthaltenen Ursprung ab. Stimmt der Ursprung nicht überein, wird der Vorgang durch das Token abgebrochen.

2.2.2 Universal Authentication Factor

Im Gegensatz zu U2F [17] bietet Universal Authentication Factor (UAF) [1] Diensten zusätzlich zur Multi-Faktor-Authentifizierung die Möglichkeit, passwortlose Authentifizierung anzubieten. Dabei werden Tokens durch Biometrie, PINs oder andere lokal bereitgestellte Mechanismen abgesichert. Das Token muss vor der erstmaligen Verwendung bei einem Dienst an den legitimen Benutzer gebunden werden. Somit kann ein UAF-Token – es sei denn, es wird ein PIN benutzt oder es können mehrere biometrische Daten hinterlegt und unterschieden werden – nur von einem Benutzer verwendet werden. Nur auf diese Weise kann das Token für passwortlose Authentifizierung genutzt werden, da das übliche, den Benutzer authentifizierende Passwort, wegfällt. Den Diensten, welche die Schnittstelle integrieren, wird dabei die Wahl gelassen, welche Art von Tokens und Mechanismen zur Authentifizierung zugelassen werden. Ein Token kann dabei durch mehrere Authentifizierungsmechanismen abgesichert werden. Dies kommt einer Multi-Faktor-Authentifizierung gegenüber dem Token gleich. Erst wenn diese erfolgreich abgeschlossen wurde, wird die benötigte Signatur durch das Token generiert.

Zur Verwendung wird, anders als bei U2F, ein authentifikatorspezifisches Modul (ASM) [50] benötigt, welches die Kommunikation zwischen dem UAF-Clienten und dem UAF-Token ermöglicht. Dabei werden Schnittstellen für verschiedene Plattformen definiert, u. a. auch für Android- und iOS-basierte Smartphones. Neben der normalen Authentifizierung kann mittels UAF [1] auch eine sicherheitslevel-basierte Authentifizierung sowie eine Transaktionsbestätigung realisiert werden. Bei der sicherheitslevelbasierten Authentifizierung wird der Benutzer für das aktuell von ihm benötigte Level sowie alle darunterliegenden Level authentifiziert. Benötigt er nun Zugriff auf ein höheres Level, wird eine erneute Authentifizierung notwendig. Je nach Level können die verwendeten Mechanismen unterschiedlich gewählt werden. Die Transaktionsbestätigung hingegen ermöglicht beispielsweise die Bestätigung und Freigabe von Überweisungen. Hierzu ist jedoch die Anzeige der Transaktion durch das Token selbst oder das ASM notwendig. Für die Transaktionsbestätigung wird neben den benötigten Metadaten von Token und Client auch die Transaktionsnachricht in die Signatur mit einbezogen.

Wie auch bei U2F, kann der Benutzer auswählen, welches angeschlossene Token für den Authentifizierungsprozess benutzt werden soll. Hierzu wird eine Liste aller angeschlossenen UAF-Tokens durch den Clienten dargestellt, aus welcher der Benutzer das

entsprechende Token wählen kann [53]. Zudem erfüllt UAF die gleichen Eigenschaften bzgl. der Privatsphäre des Benutzers wie U2F. Zusätzlich muss erwähnt werden, dass die Mechanismen zur biometrischen Authentifizierung ausschließlich lokal bereitgestellt werden. D. h. die entsprechenden biometrischen Daten des Nutzers, wie z.B. Fingerabdruck oder Gesichtsbiometrie werden lokal auf dem jeweiligen Token gespeichert und vorgehalten. Eine Speicherung oder Übermittlung der biometrischen Daten an einen FIDO-Server ist daher im Protokoll nicht vorgesehen.

2.2.3 Sicherheitsbetrachtung

Da bei Authentifizierungsprozessen die Sicherheit eine große Rolle spielt, wurden verschiedene Untersuchungen bzgl. der Sicherheit von U2F [54, 55] und UAF [56], sowie der möglichen Angriffsvektoren und Gegenmaßnahmen durchgeführt [57]. Hierbei klassifizierte die FIDO Alliance in der FIDO Security Reference [57] sechs verschiedene Angriffsklassen auf sensible Auhtentifizierungsinfrastrukturen und nennt die dazugehörigen Gegenmaßnahmen, die FIDO bietet. Diese lassen sich allgemeiner zusammenfassen in die Gruppe der Angriffe auf Server und Benutzergeräte über das Internet sowie die Gruppe der physischen Angriffe auf Benutzergeräte. Ziel ist hierbei immer das Erbeuten von Daten oder aktuellen Sessions, sodass ein Identitätsdiebstahl erfolgreich durchgeführt werden kann. Als eine der Gegenmaßnahmen wird die Verwendung von asymmetrischer Kryptographie genannt, damit auf einem FIDO-Server im besten Fall nur die öffentlichen Schlüssel der FIDO-Anmeldedaten gespeichert werden. Damit ist ein Angriff auf die Server ohne Erfolg für den Angreifer, da diese Schlüssel öffentlich bekannt sein dürfen. Zudem wird dedizierte, robuste und sichere Hardware, die sichere Elemente verwendet, für die Verwendung als Token empfohlen. Hiermit wird es dem Angreifer erschwert, die privaten Schlüssel und weiteren notwendigen Informationen aus dem Token zu extrahieren oder den Authentifikator zu kopieren.

Neben den Angriffsklassen wurden im Rahmen der FIDO Security Reference [57] 15 Schutzziele (oder Security Goal, kurz: SG) sowie weitere Gefahrenanalysen durchgeführt. Die Studie von Pereira et al. [55] zeigt dabei auf, dass weder ein korrupter Client noch ein korrupter Server ein Sicherheitsrisiko für die Protokolle darstellen. Jedoch wird ein Man-In-The-Middle-Angriff gefunden, welcher SG-3 der FIDO Security Reference gefährdet. Dieser lässt sich auf eine im Standard als optional vorgesehene Implementierung der Überprüfung des Ursprungs einer Anfrage innerhalb des Clienten zurückführen[1]. Dies kann dazu führen, dass ein böswilliger Server, welcher sich zwischen Benutzer und dem eigentlichen FIDO-Server befindet, unbemerkt Anfragen an den FIDO-Server starten kann, welche entsprechend beantwortet werden. Anschließend kann der böswillige Server die Antworten des FIDO-Servers an den Clienten unverändert weiterleiten. Dieser überprüft nicht den in den Nachrichten enthaltenen Ursprung mit dem der aktuellen Verbindung, sondern gibt die Nachrichten an das Token weiter. Das Token signiert anschließend die Daten, welche weiterhin als Ursprung den legitimen FIDO-Server enthalten. So erhält der böswillige Server eine korrekte Signatur, welche er zum

[1] Eine Implementierung dieser Überprüfung wird aus Sicherheitsgründen empfohlen.

Login gegenüber dem FIDO-Server nutzen kann. Weiterhin stellt die Studie von Chang et al. [54] Nebenkanalangriffe auf die Protokolle dar, bei denen angenommen wird, dass über die Analyse der Energieaufnahme des Tokens während der Registrierung bei einem Dienst, der geräteserienspezifische private Schlüssel sowie ein vom Hersteller auf dem Token hinterlegter tokenspezifischer privater Schlüssel zur Generierung der Key Handles ausgelesen werden können. Der Verlust des geräteserienspezifischen privaten Schlüssels könnte dabei die Vertrauenswürdigkeit ganzer Gruppen von Token angreifen, wohingegen der Verlust des tokenspezifischen privaten Schlüssels die Kommunikation des spezifischen Tokens angreifen könnte.

Auf der *Hack In The Box Security Conference*, die 2019 in Amsterdam abgehalten wurde, stellten Orru und Trotta den Revers Proxy Muraena vor [31]. Dieses Tool ermöglicht es, automatisierte Phishing-Angriffe auf Webseiten mit Zwei-Faktor-Authentifizierung durchzuführen. Dabei konnte gezeigt werden, dass viele Zwei-Faktor-Verfahren auf diese Weise angreifbar sind, bspw. die von GitHub, Dropbox und Confluence. Eine Ausnahme stellt hier U2F [17] dar, da eine Bindung des Schlüsselpaares an den jeweiligen Online-Dienst stattfindet, was den Angriff mittels eines Proxy Servers erschwert. Eine Umleitung über einen Proxy Server würde auffallen, sofern der Client die Überprüfung des Ursprungs durchführt. Dennoch konnte bereits Anfang 2018 ein anderer Angriff entwickelt werden, welches das Phishing von U2F-Tokens mittels WebUSB ermöglicht.

2.2.4 Verbreitung

Seit 2012 wurden die Standards der FIDO Alliance von vielen Firmen in ihre Authentifizierungsprozesse eingebunden [18]. So führten PayPal und Samsung bereits im Februar 2014 die passwortlose Authentifizierung auf dem Samsung Galaxy S5 ein. Auch der japanische Mobilfunkanbieter NTT DOCOMO, Facebook und Microsoft führten Standards der FIDO Alliance in ihren Produkten zur Authentifizierung ein. Dies ermöglichte die potenzielle Nutzung der Standards für über 3 Milliarden Menschen. Doch nicht nur die direkte Anwendung der Standards wurde ermöglicht, sondern auch Firmen wie Microsoft und Google implementierten native Schnittstellen zur Kommunikation zwischen Diensten und Token in ihre Betriebssysteme und Browser. Dies ermöglicht letztendlich die Mobilität der hergestellten Tokens. Zudem führten weitere Gremien, wie z.B. der Telecommunication Standardization Sector der International Telecommunication Union (ITU-T) oder das W3C, einige der Standards der FIDO Alliance als offizielle Standards in deren Kontexte ein. Dieses Kapitel geht auf die aktuellen Entwicklungen in der Nutzung und Verbreitung der Standards U2F [17], UAF [1] und WebAuthn [4] ein. Dazu werden der aktuelle Produktumfang, die Unterstützung durch Firmen sowie eine Nutzbarkeitsstudie [58] betrachtet.

Die von der FIDO Alliance bereitgestellte Zertifizierung [6] ermöglicht es Benutzern zu erkennen, ob ein Token mit den Standards der FIDO Alliance kompatibel ist und ob die Funktionalität hinsichtlich der Versprechen der Standards konform ist. Hierzu führt die FIDO Alliance eine eigene Datenbank über zertifizierte Produkte [3]. In Abbildung 2 wird die Anzahl der dort registrierten Produkte nach Standard und insgesamt dargestellt. Zudem wird die Anzahl der in der jeweiligen Produktanzahl

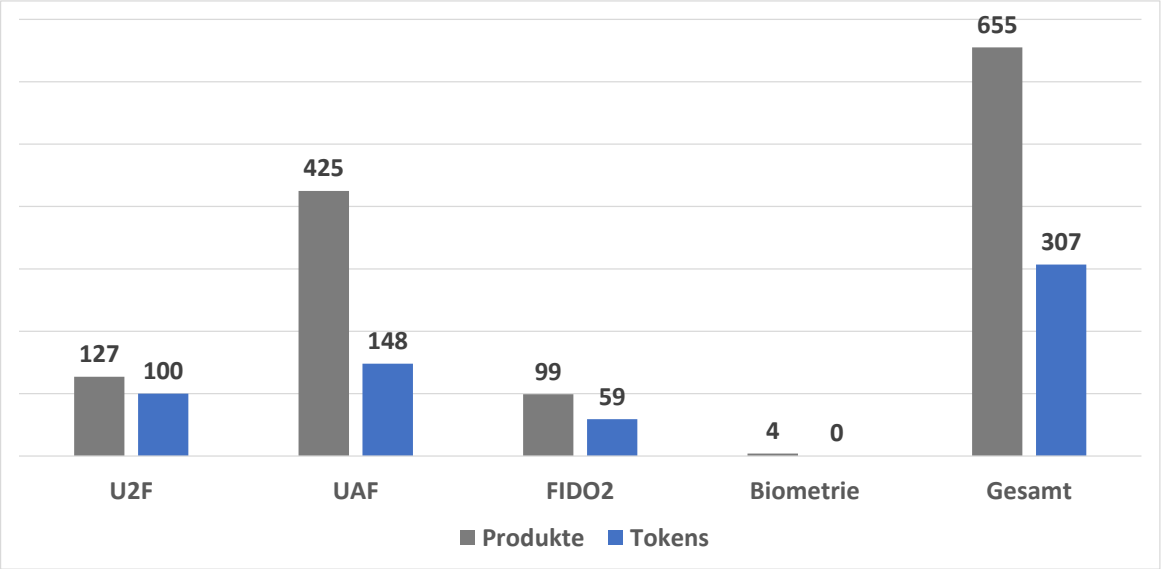

Abbildung 2: Nach Standard differenzierte Anzahl aller FIDO zertifizierten Produkte (Server, Tokens, etc.) im Vergleich zum Anteil der jeweils zertifizierten Tokens (Stand: 24.10.2019) [3].

enthaltenen Tokens aufgeschlüsselt. Neben den drei Standards zertifiziert die FIDO Alliance ebenfalls verwendete Biometriesensoren zur Verwendung in Tokens. Aus den Angaben der Datenbank (Stand: 24.10.2019) geht hervor, dass innerhalb der letzten sieben Jahre 655 Produkte zertifiziert worden sind, wovon rund 47% Tokens sind. Es wird zudem ersichtlich, dass der Großteil, ca. 65%, der Produkte auf UAF basieren. Da es keine genauen Zahlen zur Nutzung der einzelnen Standards gibt, können hier nur Vermutungen zu der Verteilung angestellt werden. Da aufgrund der in den vorherigen Kapiteln genannten Probleme jedoch nicht nur viele Bestrebungen bzgl. der Stärkung der Authentifizierung, sondern besonders auch zur Ersetzung von Passwörtern existieren, erklärt dies vermutlich das existierende Interesse. Zudem wurde der Standard z.B. durch NTT DOCOMO auf den vertriebenen Smartphones des Anbieters integriert und Organisationen wie die ITU-T führten UAF als Standardisierung innerhalb ihres Kontextes ein [18]. Vermutlich ist somit sowohl das wissenschaftliche als auch das wirtschaftliche Interesse an passwortlosen Authentifizierungsverfahren der Grund für eine hohe Anzahl an zertifizierten Produkten. Zudem erfordert die Anwendung von UAF neben Tokens und Diensten auch authentifikatorspezifische Module (ASM) zur Kommunikation mit den Clienten [50]. Auch aus diesem Umstand lässt sich schlussfolgern, dass eine größere Anzahl an UAF-Produkten zertifiziert worden ist.

FIDO2 hingegen macht nur ca. 15% der Produkte aus und ist damit auf einem ähnlichen Niveau wie der Standard U2F mit ca. 19% aller Produkte. Wird hier jedoch bedacht, dass der Standard erst seit März 2019 offizieller Standard des W3C ist [18], wird deutlich, dass großes Interesse an der Entwicklung standardkonformer Produkte besteht. Dies lässt sich vermutlich auch darauf zurückführen, dass FIDO2 mit Microsoft, Google, Samsung und anderen Firmen und Organisationen prominente Unterstützer

findet. Einen weiteren Vorteil den FIDO2 bietet, ist zudem die Abwärtskompatibilität zu U2F-zertifizierten Authentifikatoren mittels des CTAP1-Protokolls [5]. Somit können auch bereits existierende U2F-Tokens mit Diensten, die WebAuthn implementieren, für die 2FA verwendet werden. Eine Analyse[2] der in der Datenbank unter FIDO2 gelisteten Tokens und den öffentlichen auffindbaren Daten der Hersteller [3] hat ergeben, dass 19 der 59 Sicherheitsschlüssel Biometrie zur Authentifizierung verwenden. Zudem wurden insgesamt drei Authentifikatoren in Form von Smartcards entwickelt, jedoch verwendet nur eine der Smartcards biometrische Daten zur Authentifizierung. Neben den kommerziell vertriebenen Produkten zeigt auch die Open-Source-Community Interesse an der Entwicklung von Tokens. So finden sich auf GitHub bspw. das Projekt U2F Zero[3] und dessen Nachfolger Solo[4], welches eine FIDO2-Zertifizierung erhalten hat.

		WebAuthn API	
Browser	**U2F API**	**CTAP1**	**CTAP2**
Google Chrome (Desktop)	✓	✓	✓
Google Chrome (Android)	✓	✓	✓
Mozilla Firefox (Desktop)	○	✓	✓
Microsoft Edge	✓	✓	✓
Safari (macOS)	✗	○	○
Safari (iOS)	✗	✗	✗

Tabelle 1: Unterstützung der Standards U2F und WebAuthn durch gängige Browser. Es wird unterschieden in Implementiert / Stable (grün / ✓), in Entwicklung (gelb / ○) und keine Unterstützung (rot / ✗). (Stand: 23.10.2019) [8]

In Tabelle 1 wird die Unterstützung der Schnittstellen zu U2F und WebAuthn durch aktuelle Browser dargestellt [8]. Daraus geht hervor, dass Google Chrome, Mozilla Firefox und Microsoft Edge allesamt die FIDO2-Protokolle implementieren und die WebAuthn API bereitstellen. Bei den genannten Browsern fällt auf, dass die U2F API des Mozilla Firefox noch immer in Entwicklung ist. Da jedoch CTAP1 implementiert ist, reicht dies aus, um U2F-Tokens bei Diensten, die WebAuthn implementieren, zu unterstützen. Somit ist hier fraglich, ob die Implementierung vervollständigt wird. Einen Vorteil bei der Implementierung bieten hier Windows 10 und Android, da diese eine native Schnittstelle zur Kommunikation zwischen Browser und Token bieten. Somit muss kein Treiber für die Kommunikation zwischen Dienst und Token durch Browser oder andere Clienten bereitgestellt werden. Ein Token kann somit „Out-of-the-Box“ benutzt werden, sofern USB oder eines der entsprechenden Lese- bzw. Empfangsgeräte (NFC, Bluetooth LE) vorhanden sind. Der einzige Hersteller, der derzeit keine breite Unterstützung der Standards anbietet, ist Apple mit dem Browser Safari. Hier sind jedoch bereits erste Implementierungen für WebAuthn auf macOS angekündigt.

[2]Die Analyse erhebt keinen Anspruch auf Vollständigkeit. Aufgrund teilweise fehlender oder nicht erkennbarer doppelt auffindbarer Daten, kann die tatsächliche Anzahl variieren. Jedoch soll die hier getätigte Analyse eine Tendenz für die zertifizierten Produkte geben.

[3]`https://github.com/conorpp/u2f-zero` (Stand: 25.10.2019).

[4]`https://github.com/solokeys/solo` (Stand: 25.10.2019).

Trotz des großen Anklangs bei Herstellern und Dienstanbietern wurden in einer Studie [58] zur Nutzbarkeit von FIDO U2F-Tokens Probleme aufgelistet, welche die Akzeptanz der Tokens durch Benutzer verringern können. Die Studie untersuchte dabei den Umgang mit einem Sicherheitsschlüssel anhand von 61 Teilnehmern eines nichttechnischen Kurses zur Einführung in Grundlagen der Sicherheit bei Computern. Während der Studie wurden die Teilnehmer in zwei Gruppen unterteilt. Gruppe 1 musste mit einem YubiKey das offiziell vom Hersteller Yubico bereitgestellte Tutorial durchführen, während Gruppe 2 die Aufgabe bekam, den YubiKey bei ihrem Gmail-Konto einzurichten. Die auftretenden Probleme ließen sich hierbei auf falsche Annahmen, welche das Design des Tokens hervorruft, sowie auf schlecht beschriebene und unübersichtliche Prozesse zur Einrichtung der Sicherheitsschlüssel zurückführen. Dies resultierte in falschen Annahmen über die Sicherheit der Tokens sowie in Problemen bei der Anwendung und Einrichtung. Bspw. nahm ein Großteil der 27 Teilnehmer an, dass der Sensor zum Test auf Anwesenheit den Fingerabdruck des Probanden scanne und abgleiche. Auch fehlendes Feedback während der Einrichtung oder zu viele Auswahloptionen führten zur Verwirrung der Probanden. Neben den auftretenden Problemen hielt sich bis zum Schluss der Studie die Meinung unter den Teilnehmern, dass Passwörter einen ausreichenden Schutz bieten würden, da trotz der Studie kein Nutzen und keine Vorteile in der zusätzlichen Anwendung eines U2F-Tokens gesehen wurden. Nach Das et al. [58] müssen somit die Vorteile der Nutzung solcher Tokens im gesamten Prozess deutlicher hervorgehoben werden.

2.2.5 ATKey.card

Da aus [52, 59] nicht eindeutig ersichtlich wird, ob die ATKey.card eine Smartcard ist, wird, aufgrund der Ähnlichkeit in Architektur und vom Formfaktor her, davon ausgegangen, dass es sich bei diesem Produkt um eine Smartcard handelt. Daher wird im Folgenden die ATKey.card genauer beschrieben.

Die von AuthenTrend entwickelte ATKey.card ist – soweit bekannt – das einzige, zertifizierte Token mit dem Formfaktor einer Smartcard. Die Karte wird jedoch offiziell als „Badge“ bezeichnet und unterstützt die Kommunikation via USB, Bluetooth sowie NFC, ist nach Level 1 der FIDO-Zertifizierungsrichtlinien zertifiziert und verwendet einen Fingerabdrucksensor zur Authentifizierung des Nutzers. Zudem ist eine Batterie verbaut, welche sowohl für den Betrieb der Bluetooth-Antenne als auch für den Fingerabdrucksensor benötigt wird. Bei Bedarf wird die Batterie über die USB-Schnittstelle der Karte aufgeladen. Weiterhin verfügt die Karte über ein sicheres Element (SE) zur Ausführung von JavaApplets. Zusätzlich wird ein einfacher Microkontroller (MCU) zur Kommunikation zwischen SE und Fingerabdrucksensor verwendet. Die Konfiguration des Sticks wird über ein zusätzliches Tool, welches kostenlos für Windows und Mac angeboten wird, durchgeführt. Über das Tool kann z.B. die Initialisierung der Karte durch das Scannen des Fingerabdrucks vorgenommen werden. Jedoch kann die Initialisierung auch ausschließlich über die Karte erfolgen.

Im Standardumfang der Karte sind die Anwendung als FIDO U2F-Token sowie die Verwendung als NFC-Zugangskontrolle, bspw. für Türschließanlagen, enthalten.

Die FIDO2-Funktionalität wird nur bei Bedarf auf dem Token installiert und bietet die Möglichkeit einer passwortlosen Authentifikation. U2F und FIDO2 werden dabei als JavaApplets bereitgestellt und können auf alle Kommunikationsmöglichkeiten der Karte zurückgreifen. Werden die drahtlosen Kommunikationsmöglichkeiten verwendet, muss die Karte auf die interne Batterie zugreifen. Während dies bei der Anwendung von Bluetooth zum Betreiben der Antenne zwingend erforderlich ist, ist dies bei der Verwendung von NFC zum Betrieb des Fingerabdrucksensors notwendig. Damit die NFC-Funktionalität freigeschaltet wird, muss die Karte zuvor durch den Benutzer mittels Fingerabdruck freigeschaltet werden. Wurde dies erfolgreich durchgeführt, kann die Karte anschließend an ein entsprechendes NFC-Lesegerät gehalten und das entsprechende FIDO-Protokoll ausgeführt werden. Somit muss bei häufiger drahtloser Anwendung sichergestellt werden, dass die Karte regelmäßig aufgeladen wird, um die ständige Funktionalität zu gewährleisten.

3 Beitrag der Arbeit

In Kapitel 2 werden die Aktualität und die Bestrebungen bzgl. des Schutzes von Accounts ersichtlich. Es bestehen bereits viele Lösungsansätze wie Single Sign-On, Passwortmanager, Einmalpasswörter und Multi-Faktor-Authentifizierung, welche den Benutzer entlasten sollen und so den Schutz der Online-Konten erhöhen. Jedoch fehlt es vielen Verfahren derzeit an Akzeptanz aufgrund des fehlenden Sicherheitsverständnisses und des unerkannte Nutzens aus Sicht der Benutzer oder an Sicherheit gegen versierte Angreifer. Auch U2F-Token zur 2FA haben derzeit Probleme mit fehlender Akzeptanz in der Anwendung. Zudem gibt es, soweit bekannt, wenige unter den Diensten weit verbreitete Lösungen, sodass viele Dienste eigene Verfahren zur 2FA einsetzen.

Der neue Standard der FIDO Alliance [4, 5] setzt hier an, um eine einheitliche, sichere Lösung für alle Dienste zu schaffen. Wird die Schnittstelle durch einen Dienst integriert, kann jedes FIDO2- und U2F-kompatible Token mit diesem Dienst verwendet werden. Im Gegensatz zu vielen anderen Verfahren, werden die generierten Schlüssel an den jeweiligen Kontext gebunden, sodass für jeden Kontext ein eigener Schlüssel entsteht. Diese Bindung erschwert es Angreifern, die Nachrichten abzufangen und zu modifizieren. Neben dem Schutz der verschickten Daten wird bei FIDO2 zudem das Token, wie bei UAF [1], an den Benutzer gebunden. Dafür kann Biometrie oder ein PIN verwendet werden. Die Verwendung des Tokens kann dementsprechend nur durch den legitimen Benutzer ausgelöst werden. Neben den vorgesehenen Sicherheitsmaßnahmen bietet der Standard zudem einen gewissen Komfort. Je nach Implementierung stellt dieser eine passwortlose oder sogar benutzerdatenlose Authentifizierung bereit. Das heißt es fällt mindestens das klassische Passwort, welches der Benutzer für die Authentifizierung in der Regel braucht, weg. Wird die benutzerdatenlose Authentifizierung bereitgestellt, entfällt zudem auch die Eingabe des Benutzernamens.

Viele Hersteller sind an der Entwicklung von Produkten zu FIDO2 interessiert, was sich an der schnellen Veröffentlichung von Produkten seit März 2019 zeigt [3]. Dies spiegelt ebenfalls die Aktualität des FIDO2-Standards und der zu lösenden

Problematik wieder. Neben USB-Sicherheitsschlüsseln und Smartphone-Apps sind auch weitere Formfaktoren als Tokens verfügbar. Darunter fällt z.B. die im vorherigen Kapitel vorgestellte ATKey.card [52] von AuthenTrend, welche den Formfaktor einer Smartcard besitzt und den Benutzer per Fingerabdruck authentifiziert. Die Karte selbst ist nach FIDO2 Level 1 zertifiziert und unterstützt alle für FIDO2 vorgesehenen Kommunikationsprotokolle.

Im Rahmen dieser Arbeit wurde ein Token in Form einer Smartcard entwickelt, welches ausschließlich durch die vom Leser während der Benutzung bereitgestellte Energie versorgt wird und die passwortlose Authentifizierung mittels Fingerabdruck und FIDO2 über NFC ermöglicht. Die Struktur des Tokens ähnelt dabei dem der ATKey.card, jedoch erfolgt die Stromzufuhr ausschließlich über den vom NFC-Lesegerät induzierten Strom. Optional kann zudem anstelle des Fingerabdrucks die von FIDO2 vorgesehene Client-Pin-Methode zur Authentifizierung angewendet werden. Die FIDO2-Funktionalität wurde dabei als Java Card 3.0.2 Applet implementiert und liegt innerhalb eines sicheren Elements, welches auf der Smartcard verbaut wurde. Es ist damit – soweit bekannt – das erste Token, welches Biometrie zur Authentifizierung ohne internes Batteriepack oder direkte Verbindung per USB ermöglicht. Dies zeigt, dass FIDO2 auch unter herausfordernden Bedingungen, wie fehlender interner Batterie und geringer Speicherkapazität, mittels Biometrie angewendet werden kann. Das in dieser Arbeit entwickelte Token wurde zudem auf seine Zertifizierbarkeit hin untersucht. Die Erkenntnisse der Untersuchung werden in dieser Arbeit dargestellt und ausstehende Maßnahmen für eine erfolgreiche Zertifizierung diskutiert.

Neben der Implementierung des Java Card Applets konnten während des Entwicklungsprozesses Unzulänglichkeiten und Fehler in dem FIDO2-Standard der FIDO Alliance sowie der im Internet verfügbaren Testseiten festgestellt werden. Zudem wurden Probleme gefunden, welche die Nutzbarkeit von FIDO2 ggf. einschränken.

4 FIDO2

Im Februar 2016 begannen die Bestrebungen des W3C und der FIDO Alliance, zusammen eine einheitliche Lösung für die sichere Authentifikation bei Online-Diensten auf Grundlage der FIDO2 API zu standardisieren [18]. Dies führte im April 2018 zur offiziellen Bekanntgabe der FIDO2-Spezifikation, welche zugleich als Standard vorgeschlagen wurde. Die zu FIDO2 dazugehörige Web-Authentication-Spezifikation [4] wurde schließlich im März 2019 zum offiziellen Standard des W3C, welcher die in den FIDO2-Spezifikationen definierten Client-to-Authenticator-Protokolle (CTAP) [5] verwendet. Mittlerweile bieten Android und Microsoft native Schnittstellen zur Kommunikation mit FIDO2-Tokens über USB, NFC und Bluetooth an. Zudem wurde die FIDO2 API in die Browser Mozilla Firefox, Google Chrome und Microsoft Edge integriert [8]. Hierbei greifen die Browser auf die von den zuvor genannten Betriebssystemen bereitgestellte API zur Kommunikation mit den Tokens zu. Auch Apple arbeitet bereits an einer Unterstützung der Standards durch Safari.

Ziel von WebAuthn [4] und den CTAP-Protokollen [5] ist es, eine einheitliche Schnittstelle zur sicheren Authentifikation bei Online-Diensten bereitzustellen und die Kommunikation zwischen Dienst und Token zu spezifizieren. Dies soll die Anwendung eines FIDO2-Tokens bei jedem Online-Dienst, welcher FIDO2 in seinen Authentifikationsprozess integriert, ohne zusätzliche Treiber oder Clienten ermöglichen. Weiterhin wird asymmetrische Kryptographie zur Erzeugung von eindeutigen Schlüsselpaaren zwischen dem Token und den einzelnen Diensten angewendet. Die generierten Schlüsselpaare werden dabei an den Authentifikator und den Dienst gebunden. Während mit U2F [17] und UAF [1] zwei eigenständige, miteinander inkompatible Schnittstellen spezifiziert wurden, welche je einen eigenen Zweck verfolgen, werden mit FIDO2 beide Anwendungsfälle – 2FA und passwortlose Authentifizierung – über eine gemeinsame Schnittstelle ermöglicht. Dabei wird die 2FA über das CTAP1 / U2F-Protokoll definiert, welches eine Kompatibilität von WebAuthn mit U2F-Tokens vorsieht. Somit wird für die 2FA über WebAuthn nicht zwingend ein FIDO2-Token benötigt, stattdessen können ebenfalls bereits existierende U2F-Tokens verwendet werden.

Zusätzlich bietet FIDO2 die Möglichkeit der benutzerdatenlosen Authentifizierung an. Hierbei unterscheidet sich die benutzerdatenlose von der passwortlosen Authentifizierung darin, dass der Benutzer für die Anmeldung bei einem Dienst weder das Passwort noch den Benutzernamen angeben muss. Dazu muss der Benutzer die Authentifizierung mittels FIDO2 starten und anschließend das für den Account hinterlegte Token aktivieren. Um die passwort- oder benutzerdatenlose Authentifizierung nutzen zu können, muss das FIDO2-Token zuvor an den Benutzer gebunden werden. FIDO2 sieht dazu die Anwendung von Biometrie oder einer durch den Nutzer wählbaren PIN vor. Den Ablauf der Kommunikation definiert dabei das CTAP2-Protokoll.

Im Folgenden werden der zu FIDO2 gehörige Standard WebAuthn und die ebenfalls dazugehörigen CTAP-Protokolle ausführlich beschrieben.

4.1 Web Authentication

Der Standard Web Authentication [4] beschreibt die Schnittstellen zum Erstellen und zur Anwendung von Schlüsselpaaren, welche für FIDO2 benötigt werden. Dazu werden Datenstrukturen, Schnittstellen, Algorithmen, Modelle und weitere Verfahren definiert, welche für die Implementierung der Schnittstelle notwendig sind. Das Dokument spricht dabei verschiedene Entwicklergruppen an – diese sind Entwickler von Webdiensten, Frameworks, Clienten, Betriebssystemen sowie Authentifikatoren.

In diesem Kapitel wird die in WebAuthn definierte Schnittstelle beschrieben sowie auf das von WebAuthn erwartete Authentifikatormodell eingegangen. Zudem wird das Vertrauensmodell für Authentifikatoren und die Generierung der Signaturen betrachtet. Daraufhin erfolgt eine Sicherheitsbetrachtung sowie abschließend eine Analyse zum Schutz der Privatsphäre.

4.1.1 Schnittstelle

Die Kommunikation zwischen Client (z.B. Browser) und Dienst findet über die WebAuthn API [4] statt. Diese überträgt die notwendigen Daten zwischen den beiden Kommunikationspartnern und ermöglicht so die Erstellung und Verwendung von FIDO2-Schlüsselpaaren durch den Nutzer. Der allgemeine Ablauf gleicht dabei dem in Kapitel 2.2 in Abbildung 1 für U2F und UAF dargestellten Verlauf. FIDO2 ermöglicht dabei ebenfalls die Einbindung einer PKI, wie zum Beispiel des FIDO-Metadaten-Services, zur Verifikation des im Token hinterlegten Zertifikats und der Eigenschaften des Tokens. Zudem wird der zum Zertifikat dazugehörige private Schlüssel auf dem Token hinterlegt. Das Vertrauen in verschiedene Zertifikate kann hierbei durch den jeweiligen Dienst konfiguriert werden und somit die Verwendung bestimmter Tokens zugelassen werden. Weiterhin besteht, im Gegensatz zu UAF, keine Möglichkeit mehr, direkte Richtlinien über zu verwendende Mechanismen für die Benutzerauthentifizierung festzulegen. Stattdessen wird hier von den spezifischen Mechanismen, wie z.B. Fingerabdruck, Retinascan oder Gesichtsbiometrie, abstrahiert und nur noch darin unterschieden, ob eine Benutzerauthentifizierung durch das Token möglich ist oder nicht.

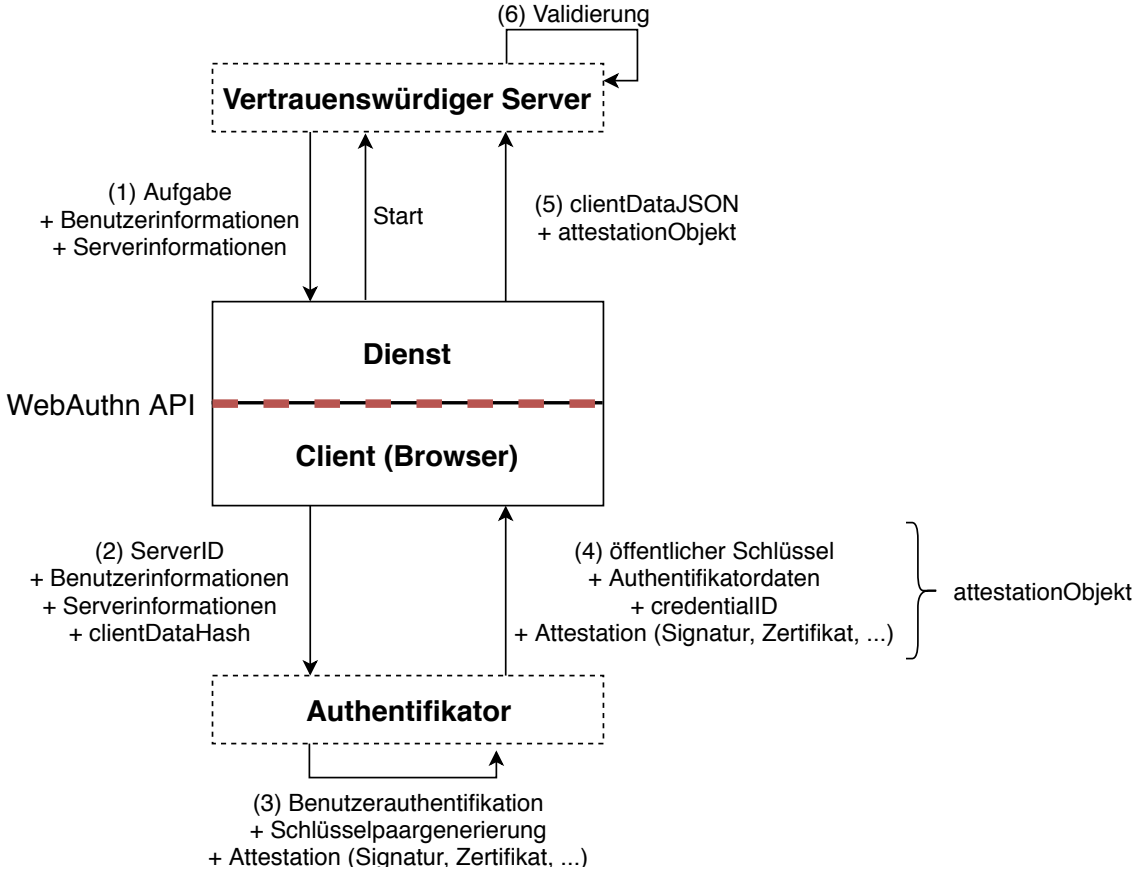

Abbildung 3: Ablauf und Funktionsweise der WebAuthn API während der Registrierung eines FIDO2-Tokens nach [4]. Die Schnittstelle ist dabei für die Kommunikation zwischen Client und Dienst zuständig.

In Abbildung 3 werden die für die Registrierung eines FIDO2-Tokens ausgetauschten Informationen dargestellt. Hierbei werden die vom Server übermittelten Daten (1) durch den Clienten um den *clientDataHash* und die *ServerID* erweitert. Der Hash basiert u.a. auf der vom Server bereitgestellten *Aufgabe* und den vom Clienten erkannten Ursprung der Nachricht. Diese Daten werden nun an das Token übermittelt (2). Konnte sich der Benutzer erfolgreich gegenüber dem Token authentifizieren, wird das neue Schlüsselpaar generiert (3). Zur Bindung des Schlüsselpaares an den Dienst wird nun eine credentialID angelegt, welche dem Server übermittelt wird. Zudem wird eine Signatur über einige Daten des Authentifikators, die sogenannten *authenticatorData*, sowie den *clientDataHash* generiert. Zur Berechnung der Signatur wird ein auf dem Token während der Herstellung hinterlegter, zum Zertifikat passender, privater Schlüssel verwendet. Anschließend werden der für den Dienst generierte öffentliche Schlüssel, die vom Token generierte *credentialID*, die *Auhentifikatordaten*, die *Signatur* und das *Zertifikat* des Tokens an den Clienten geschickt (4). Dieser übermittelt die zur Generierung des clientDataHashes benutzten *clientData*, samt der vom Token übermittelten Daten an den Server (5). Der Server kann nun aus den vorliegenden Daten den *clientDataHash* berechnen. Anschließend kann der Server mittels des im Zertifikat hinterlegten öffentlichen Schlüssels die Signatur validieren und die signierten Daten verifizieren (6). Stimmen diese überein, kann auf eine korrekte Kommunikation geschlossen werden und das Schlüsselpaar wurde erfolgreich etabliert.

Soll eine Authentifizierung, wie in Abbildung 4 dargestellt, durchgeführt werden, generiert der Server eine neue *Aufgabe* und übermittelt diese an den Clienten (1). Gegebenenfalls werden die für den Account hinterlegten *credentialIDs* ebenfalls übermittelt. Dies ist zum Beispiel notwendig, wenn das Token den privaten Schlüssel zur Speicherung direkt verschlüsselt oder Material zur deterministischen Schlüsselableitung in die *credentialID* auslagert. Dies kann auch notwendig sein, wenn der Server nur bestimmte *credentialIDs* zur Authentifizierung zulassen möchte. Anschließend wird die *Aufgabe* in die Generierung des *clientDataHash* einbezogen. Dieser wird samt *ServerID* vom Clienten an das Token gesendet (2). Konnte sich der Benutzer – wie bei der Registrierung – erfolgreich gegenüber dem Token authentifizieren, wird nun eine Signatur mit dem privaten Schlüssel des für den Dienst etablierten Schlüsselpaares über die vom Token ermittelten *Authentifikatordaten* und den *clientDataHash* generiert (3). Anschließend werden die verwendeten *Authentifikatordaten* sowie die generierte Signatur an den Clienten übermittelt (4). Dieser übermittelt nun die vom Token generierten Daten sowie die zur Generierung des *clientDataHash* verwendeten *clientData* an den Server (5). Auch hier kann der Server nun die Signatur mittels des für den Dienst etablierten Schlüsselpaares validieren (6). Stimmen die vorliegenden Daten mit denen des Tokens überein, kann ebenfalls darauf geschlossen werden, dass das korrekte Schlüsselpaar und die korrekten Daten zur Generierung der Signatur verwendet wurden. Die Authentifizierung durch den Benutzer ist somit erfolgreich.

Während der Verwendung der Schnittstelle wird die Generierung einer Signatur notwendig, je nach Fall sogar die Generierung eines Schlüsselpaares. Hierzu sind im Standard die verschiedenen unter CBOR Object Signing and Encryption (COSE) [60] registrierten Verfahren und Algorithmen vorgesehen. Zudem definiert WebAuthn

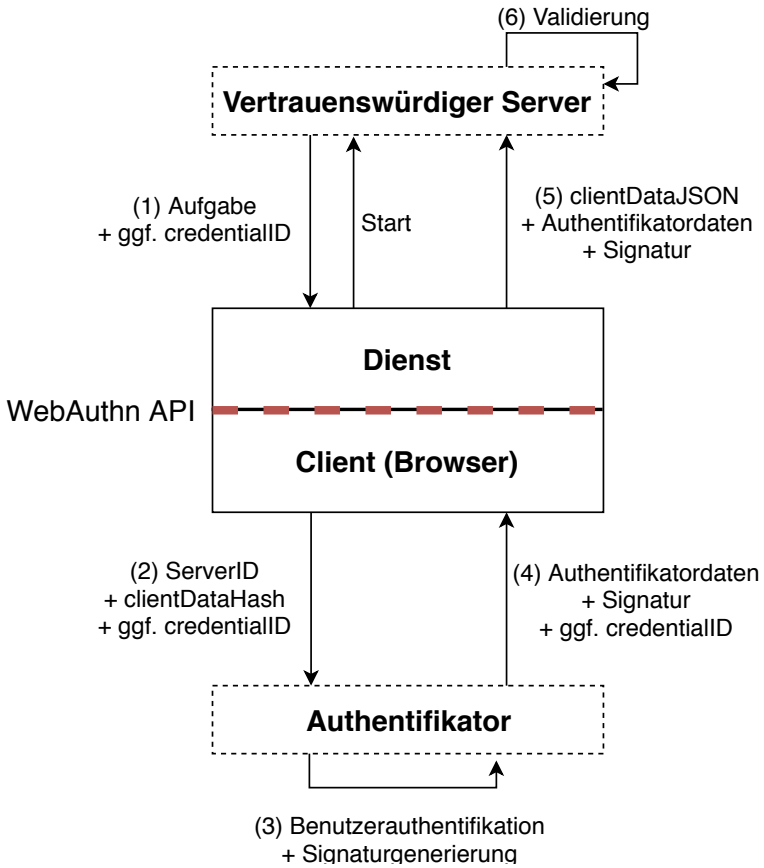

Abbildung 4: Ablauf und Funktionsweise der WebAuthn API während der Authentifizierung mittels eines FIDO2-Tokens nach [4]. Die Schnittstelle ist dabei für die Kommunikation zwischen Client und Dienst zuständig.

selbst einige Verfahren, welche ebenfalls zu den unter COSE gelisteten Verfahren zur Verwendung hinzugefügt wurden [61]. Die dort definierten Verfahren basieren auf RSA oder Elliptischen Kurven. Das zu verwendende Verfahren wird dabei während der Registrierung des Tokens bei einem Dienst festgelegt. Der Server kann hierbei die von ihm unterstützten Verfahren nach Präferenz sortieren und direkt in Schritt (1) innerhalb der Serverinformationen als Liste an das Token senden. Das Token durchläuft die Liste anschließend und verwendet das erste von ihm unterstützte Verfahren zur Generierung des Schlüsselpaares und der Signatur. Wird keines der Verfahren durch das Token unterstützt, wird die Registrierung abgebrochen [5]. Konnte wiederum ein Schlüsselpaar erfolgreich etabliert werden, wird das festgelegte Verfahren fortan bei jeder Authentifizierung gegenüber dem Dienst angewendet.

In Abbildung 3 und 4 wird nur der Austausch der wichtigen Daten zur Registrierung und Authentifizierung dargestellt. Das Protokoll [4, 5] bietet jedoch dem vertrauenswürdigen Server die Möglichkeit, die ausgetauschten Nachrichten um weitere Informationen zu erweitern. Diese Erweiterungen ermöglichen es dem Server, Anpassungen für bestimmte Anwendungsfälle vorzunehmen. Dabei können Erweiterungen sowohl die

Arbeitsweise des Clienten als auch die des Authentifikators beeinflussen oder besondere Informationen einfordern. Unbekannte Erweiterungen werden hierbei vom Clienten ignoriert und ggf. an das Token weitergereicht. Ist die Erweiterung auch dem Token unbekannt, ignoriert es diese ebenfalls. Alle Erweiterungen müssen so implementiert werden, dass diese weder die Sicherheit noch die Privatsphäre des Benutzers gefährden. Dabei können sowohl eigene Erweiterungen implementiert als auch die im Standard definierten Erweiterungen verwendet werden – hierzu zählt zum Beispiel die Darstellung von Transaktionstexten, die an das Token zur Freigabe übermittelt werden sollen.

4.1.2 Authentifikatoren

Neben der Schnittstelle zwischen Client und Dienst definiert der WebAuthn-Standard [4] ein Authentifikatormodell, welches die Erwartungen des Clienten und die Möglichkeiten zur Gestaltung eines FIDO2-Tokens beschreibt. Jeder Authentifikator hat dabei eine eindeutige AAGUID zur Identifikation der Geräteserie, welche durch den Hersteller des Tokens gewählt wird. Die Hauptfunktionalität des Tokens besteht darin, Signaturen zu erzeugen, welche während der Registrierung und der Authentifikation benötigt werden. Die Signaturen werden an einen Kontext gebunden, welcher durch den Clienten, den Server und den Authentifikator selbst definiert ist. Anschließend wird die Signatur durch den privaten Schlüssel des mit dem Dienst etablierten oder zu etablierenden Schlüsselpaares signiert. Dies soll die Integrität der erstellten Signatur wahren. Während der Authentifikation wird die vom Token erstellte Signatur durch den Server geprüft, indem dieser die Signatur auf denselben Daten verifiziert. Stimmt diese mit der vom Sicherheitsschlüssel generierten Signatur überein, kann von der Integrität der Daten und einem legitimen Kontext ausgegangen werden. Die jeweilige Operation – Registrierung oder Authentifikation – ist erfolgreich.

Eigenschaft	**Optionen**	
Mobilität / Anwendbarkeit	integriert	mobil
Speicherung der Schlüsselinformationen	intern	serverseitig
Authentifikationsmöglichkeiten	Ein-Faktor	Multi-Faktor

Tabelle 2: Mögliche Eigenschaften eines Authentifikators nach [4].

Authentifikatoren können in den verschiedensten Formen hergestellt werden und so die Bedürfnisse unterschiedlicher Anwender erfüllen. Dabei bietet jede mögliche Kombination an Eigenschaften unterschiedliche Vor- und Nachteile. Die Tabelle 2 listet die jeweiligen Optionen bezüglich der drei definierenden Eigenschaften – Mobilität, Speicherung und Authentifkation – beim Design eines Tokens auf. Das mögliche Design reicht von einem fest integrierten Plattformauthentifikator, mit eigenem Speicher, der nur Ein-

Faktor-Möglichkeiten besitzt, bis hin zu einem mobilen Token, welches keinen eigenen Speicher besitzt, jedoch Multi-Faktor-Authentifizierung anbietet. Hierbei beschreiben die einzelnen Eigenschaften den Anwendungsbereich eines Tokens. Die nachfolgende Auflistung beschreibt die Optionen der einzelnen Eigenschaften detaillierter.

Mobilität Fest in ein Endgerät integrierte Token können nur mit der entsprechenden Plattform interagieren, wie zum Beispiel ein in einen PC fest integrierter Hardware-Authentifikator. Ein fest verbauter Authentifikator ist in der Regel nicht flexibel und ermöglicht die Authentifikation somit nur am entsprechend dazugehörigen Endgerät. Mobile Authentifikatoren können dagegen potenziell mit jedem Endgerät verwendet werden, sofern entsprechende Kommunikationsmöglichkeiten zwischen Token und der Plattform vorhanden sind. Dies ermöglicht eine große Flexibilität in der Nutzung, jedoch muss ein mobiler Authentifikator immer separat mitgeführt werden, um sich jederzeit von überall authentifizieren zu können. Eine Ausnahme stellen z.B. Smartphones dar. Es kann vorkommen, dass der Authentifikator innerhalb des Smartphones fest verbaut ist, der Authentifikator jedoch auf die Bluetooth- oder NFC-Schnittstelle des Handys zugreifen darf und so gegenüber anderen Endgeräten als mobiles Token fungieren kann. Somit treffen beide Eigenschaften auf einen Authentifikator zu und die Anwendungsweise kann durch den Benutzer selbst bestimmt werden.

Speicherung Wird die Speicherung der Daten betrachtet, kann auch hier aus verschiedenen Vorgehensweisen gewählt werden. Zum einen kann die Speicherung der privaten Schlüssel auf dem Token selbst erfolgen, zum anderen auf Seiten des Dienstes. Wird eine Speicherung intern auf dem Token angeboten, kann hier – je nach Speichertyp, z.B. innerhalb eines sicheren Elements (SE) – ein unterschiedlicher Schutz der privaten Schlüssel forciert werden. Dabei wird zwischen einem sicheren und einem normalen Speicher unterschieden. Wird ein normaler Speicher verwendet, so müssen die dort hinterlegten privaten Schlüssel durch eine geeignete Verschlüsselung geschützt werden. Zudem können die privaten Schlüssel nur durch den physischen Verlust des Tokens verloren gehen. Jedoch wird durch die Verwendung eines internen Speichers die potentielle Anzahl an Diensten, mit denen das Token verwendet werden kann, eingeschränkt. Ist der Speicher voll, können keine weiteren Schlüsselpaare für neue Dienste generiert werden. Diese Problematik ist nicht gegeben, wenn die Speicherung auf Seiten des Dienstes erfolgt. In diesem Fall kann das Token mit beliebig vielen Diensten verwendet werden. Jedoch muss vor der Benutzung der private Schlüssel vom Server an das Token übermittelt werden. Weiterhin erfordert dieses Vorgehen einen separaten Schutz des vom Token für den Dienst generierten privaten Schlüssels, so dass nur das legitime Token den privaten Schlüssel wieder extrahieren kann. Auch hier ist eine Kombination beider Optionen möglich, sodass – je nach Dienst – eine Speicherung intern oder serverseitig erfolgen kann.

Authentifikation Die Authentifikationsmöglichkeit bezieht sich auf die in 2.1 dargestellten Kategorien, denen Authentifikatoren zugeordnet werden können. Das

Token selbst wird dabei in die Kategorie „Besitz“ eingeordnet. Bietet das Token ein weiteres Verfahren einer anderen Kategorie zur Authentifizierung des Nutzers an, z.B. einen Fingerabdruckscan, so ist das Token selbst zur Multi-Faktor-Authentifizierung geeignet und kann somit zur passwort- oder benutzerdatenlosen Authentifikation benutzt werden. Wird hingegen kein weiteres Verfahren zur Authentifizierung des Benutzers angeboten, kann das Vorhandensein des Tokens selbst nur als Anwesenheit eines Benutzers gewertet und somit nur zur 2FA verwendet werden. Die Anwesenheit eines Benutzers könnte beispielsweise durch das Betätigen eines auf dem Token befindlichen Knopfes zusätzlich bestätigt werden. Das Token stellt somit nur einen zusätzlichen Faktor des Typs „Besitz“ dar. Hierbei impliziert die Authentifikation des Nutzers anhand eines weiteren Faktors, wie den Fingerabdruck, die Anwesenheit des Benutzers. Die Anwesenheit eines Benutzers impliziert jedoch nicht die Legitimität des Benutzers. Dabei teilt das Token die erfolgte Prüfung der Anwesenheit und ggf. die durchgeführte Authentifikation dem Server mit.

Die MFA des Benutzers gegenüber dem Token kann hierbei vielfältig gestaltet werden [4]. Neben biometrischen Sensoren oder anderen Eingabemöglichkeiten, wie die direkt PIN-Eingabe auf dem Token, bietet der WebAuthn-Standard auch die Verwendung einer *clientPin* an. Diese ist in die CTAP-Protokolle integriert [5]. Die PIN muss mindestens 4 Zeichen lang sein und sollte eine minimale Maximallänge von 63 Zeichen aufweisen. Die *clientPin* wird dabei über den Clienten auf dem Token etabliert und kann anschließend zur Benutzerauthentifizierung gegenüber dem Token genutzt werden. Dabei sollte das Token nach maximal acht aufeinanderfolgenden falschen Eingaben gesperrt werden.

Neben den drei vorgestellten Eigenschaften, kann ein Authentifikator dem Benutzer verschiedene Konfigurationsmöglichkeiten bieten – z.B. die Initialisierung der verschiedenen biometrischen Sensoren oder die Auswahl der oder des anzuwendenden biometrischen Verfahren(s). Hierfür kann eine vom Hersteller entwickelte Software zur Einrichtung des Tokens bereitgestellt werden. Auch der Werksreset oder das Verwalten einzelner Schlüsselpaare sollte über dieses Tool ermöglicht werden. Dies ist auf die unterschiedlichen Implementierungsweisen der Tokens zurückzuführen und daher herstellerabhängig [4].

4.1.3 Vertrauensmodell

Nachfolgend werden die von WebAuthn [4] definierten Vertrauensmodelle betrachtet. Anhand des verwendeten Verfahrens kann der Server sein Vertrauen in die Integrität der Daten und das Token selbst festlegen. Der Dienst ermittelt dabei das Vertrauen während des Registrierungsprozesses anhand des in der Antwort des Tokens enthaltenen *attestationObjekts*. Dazu ermöglichen die Vertrauensmodelle die Stärke der Attestation sowie die vom Authentifikator angegebenen Eigenschaften, wie z.B. die Möglichkeit der Benutzerauthentifizierung mittels Token, zu beurteilen. Je nach verwendetem Verfahren umfasst das *attestationObjekt* neben den Authentifikatordaten und der Signatur entweder ein Zertifikat oder andere Informationen zur Verifizierung der Angaben des Tokens.

Die Beurteilung kann beispielsweise auch mit Hilfe des FIDO-Metadaten-Services unterstützt werden. Die Einschätzung der Vertrauenswürdigkeit findet dabei während des Registrierungsprozesses eines Tokens gegenüber einem Dienst statt. Stimmen anschließend die für die Überprüfung ausgetauschten Daten mit den Berechnungen und Daten des Servers überein und konnte aufgrund weiterer Informationen, wie z.B. des Zertifikats, die Vertrauenswürdigkeit des Tokens festgestellt werden, ist die Registrierung erfolgreich. Das Token kann nun mit dem Dienst verwendet werden. Da während der Registrierung das Vertrauen zwischen Dienst und Token hergestellt wurde, muss dies nicht noch einmal geprüft werden. Ab jetzt wird zur Authentifizierung das während der Registrierung etablierte Schlüsselpaar zur Erstellung der Signatur verwendet.

Im Folgenden werden die vier verschiedenen Vertrauensmodelle *Basic Attestation*, *Self Attestation*, *Attestation CA* und *Elliptic Curve based Direct Anonymous Attestation* dargestellt.

Basic Attestation Die Basic Attestation [4, 53] basiert auf einem Schlüsselpaar und einem für das Schlüsselpaar ausgestellten Zertifikat, welche spezifisch für eine Geräteserie generiert worden sind. Dabei werden der private Schlüssel des Schlüsselpaares sowie das Zertifikat auf dem Authentifikator hinterlegt. Hierbei stellt das Zertifikat die Vertrauensgrundlage des Authentifikators dar – anhand der Zertifikatskette, die über das Zertifikat ermittelt werden kann, kann so die Vertrauenswürdigkeit des Tokens beurteilt werden. Zusätzlich wird die vom Token übermittelte Signatur mit dem zum Zertifikat gehörigen privaten Schlüssel erstellt. Mit dem für das Zertifikat hinterlegten öffentlichen Schlüssel kann vom Server nun die Authentizität der Daten beurteilt werden. Dieses Vertrauensmodell wird auch für die in Kapitel 4.1.1 gegebenen Beispiele verwendet.

Self Attestation Kann ein Token den Schutz eines privaten Schlüssels, wie er für die Basic Attestation benötigt wird, nicht garantieren, muss es auf die Self Attestation zurückgreifen [4, 53]. In diesem Fall wird kein für das Modell spezifisches Schlüsselpaar etabliert, sondern auf das während der Registrierung generierte dienstspezifische Schlüsselpaar zurückgegriffen. Das Token signiert nun die für die Signatur benötigten Daten mittels des dienstpezifischen privaten Schlüssels. Anstelle des Zertifikats muss bei der Self Attestation ein anderes Verfahren zur Bestätigung der Eigenschaften des Tokens herangezogen werden. Dazu kann z.B. der FIDO-Metadaten-Service genutzt werden.

Attestation CA Das Modell der Attestation CA [4] basiert auf der Existenz eines Trusted Platform Modules (TPM) innerhalb des Authentifikators. Dabei besitzt jedes TPM einen für das TPM, und damit auch für den Authentifikator, individuellen Endorsement Schlüssel. Dieser Schlüssel ermöglicht es dem Token, gegenüber einem vertrauenswürdigen Dritten in der Rolle eines Zertifikatsausstellers, während eines Registrierungsvorgangs ein neues Schlüsselpaar und das dazugehörige Zertifikat anzufordern. Ein neues Zertifikat ersetzt dabei die vorherigen Zertifikate des Tokens. Somit kann die Gültigkeit der Zertifikate und Schlüsselpaare dynamisch

angepasst werden. Es besteht sogar die Möglichkeit, für jeden Registrierungsprozess ein neues Zertifikat auszustellen und das dazugehörige Schlüsselpaar generieren zu lassen.

Elliptic Curve based Direct Anonymous Attestation Die Elliptic Curve based Direct Anonymous Attestation (ECDAA) [4, 53, 62, 63] etabliert auf dem Token während der ersten Nutzung oder Personalisierung ein spezifisches Schlüsselpaar, basierend auf elliptischen Kurven. Auf jedem Token wird dazu während der Herstellung ein eigenes, spezifisches Geheimnis hinterlegt. Anschließend wird dieses Geheimnis während der erstmaligen Nutzung oder Personalisierung an einen vertrauenswürdigen Dritten gesendet, den ECDAA-Aussteller. Dieser generiert nun ein auf elliptischen Kurven basierendes, eigenes Schlüsselpaar für das Token. Der private Schlüssel wird auf dem Token hinterlegt und für die Generierung der Signatur verwendet. Dabei dient die erstellte Signatur als Zero-Knowledege-Beweis dafür, dass ein vertrauenswürdiger Authentifikator zur Erstellung der Signatur verwendet wurde. Weiterhin wird der öffentliche Schlüssel des Schlüsselpaares dem Server zugänglich gemacht, der Zugriff kann dabei z.B. über den FIDO-Metadaten-Service gestaltet werden.

Ist keines der Vertrauensmodelle für das Token implementiert, kann dieses dennoch zur Registrierung verwendet werden [4]. In diesem Fall wird jedoch weder eine Signatur noch ein Zertifikat zur Validierung der Daten bereitgestellt. Die Daten sind somit nicht integritätsgeschützt und auch die vom Token angegebenen Eigenschaften können nicht bestätigt werden. Daher muss jeder Dienst das Vertrauen in solche Tokens durch dienstspezifische Richtlinien festlegen. Gleiches gilt im Falle der Self Attestation, da bei diesem Verfahren kein durch ein Zertifikat etabliertes Schlüsselpaar, sondern zum Zeitpunkt der Registrierung ein durch das Token generiertes Schlüsselpaar für die Erstellung der Signatur verwendet wird.

4.1.4 Signaturen

Die in WebAuthn [4] beschriebenen Signaturen dienen nicht nur zur Ermittlung des Vertrauens in ein Token während des Registrierungsprozesses. Weiterhin werden die Signaturen zur Bindung des dienstpezifischen Schlüsselpaares an den aktuell geltenden Kontext und zur Wahrung der Integrität der während des Registrierungs- oder Authentifikationsprozesses übermittelten Daten verwendet. Die Bindung an den Kontext wird dadurch erzielt, dass in die Generierung einer Signatur sowohl der *clientDataHash* als auch bestimmte *Authentifikatordaten* einfließen.

Wie in Abbildung 5 dargestellt, werden zur Erstellung der Signatur zuerst die *Authentifikatordaten* sowie der vom Clienten übertragene *clientDataHash* konkateniert. Daraufhin findet die Signierung der konkatenierten Daten statt. Der verwendete private Schlüssel unterscheidet sich dabei je nach ausgeführter Operation. Wird die Signatur während der Registrierung eines Tokens bei einem Dienst erstellt, wird der durch das Vertrauensmodell bereitgestellte private Schlüssel verwendet. Soll die Signatur zur

Authentifizierung dienen, wird der zu dem Dienst hinterlegte private Schlüssel des zuvor etablierten Schlüsselpaares benutzt.

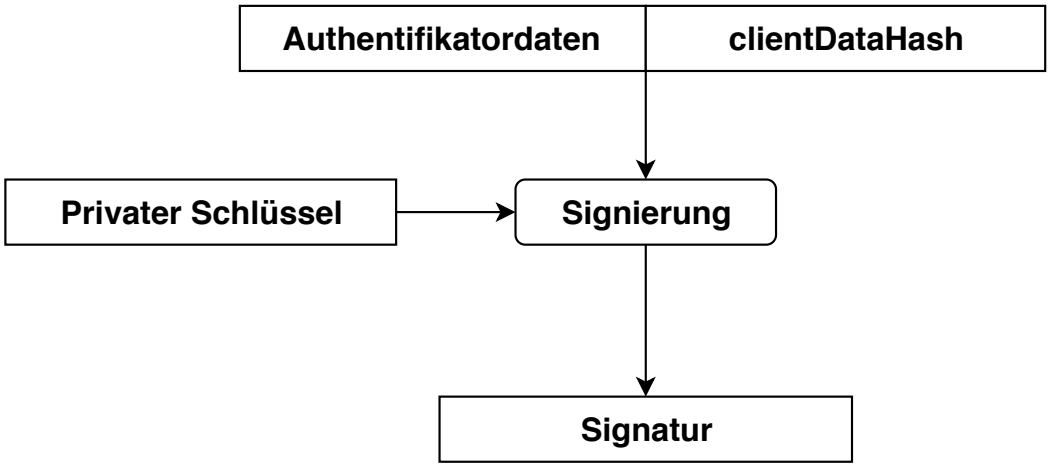

Abbildung 5: Erstellung der Signatur für WebAuthn nach [4]. Während der Registrierungsphase wird der private Schlüssel anhand des verwendeten Vertrauensmodells gewählt. Bei der Authentifizierung gegenüber einem Dienst wird der private Schlüssel des dienstspezifischen Schlüsselpaares verwendet.

Feld	Beschreibung
type	Beschreibt den aktuell ausgeführten Vorgang. Bei Registrierung der String „webauthn.create“ bei Authentifizierung „webauthn.get“.
challenge	Die vom Server übermittelte Aufgabe.
origin	Ursprung des Servers, aus Sicht des Clienten ermittelt.
tokenBinding	Optionales Feld. Angaben zum TLS-Token-Binding, werden vom Clienten erhoben.

Tabelle 3: Darstellung der Datenstruktur *clientData* nach [4].

Die Tabellen 3 und 4 zeigen dabei die Informationen, welche zur Generierung der Signatur benötigt werden. Es wird ersichtlich, dass keine persönlichen Daten des Benutzers zur Authentifizierung ausgetauscht werden müssen. Zudem wird durch das Verwenden verschiedener veränderlicher Daten, wie z.B. die *challenge* in den *clientData* oder dem *signCount* der *Authentifikatordaten*, gewährleistet, dass keine generierte Signatur einer bereits zuvor erstellten oder nachfolgenden Signatur gleicht. Dies erschwert die erfolgreiche Durchführung von Man-in-the-Middle- oder Replay-Angriffen. Zudem bietet der *signCount* des Tokens einen Schutz gegen das Kopieren eines Authentifikators durch einen Angreifer. Der *signCount* kann zusätzlich nach

jeder ausgeführten Operation auf dem Server gespeichert werden. Wird nun ein Token kopiert, sodass diese Kopie zur Authentifizierung und Registrierung durch den Angreifer benutzt werden kann, würde dies den hinterlegten *signCount* bei jeder Nutzung erhöhen. Würde nun das Original durch den legitimen Benutzer verwendet werden, würde der Server anschließend den kleineren *signCount* bemerken und so darauf schließen können, dass zwei identische Tokens im Einsatz sind.

Feld	Beschreibung
rpIdHash	Hash über die an das Token übermittelte ServerID.
flags	Acht Flags zur Beschreibung der Authentifikatordaten. Insgesamt ein Byte lang. 1 steht für true, 0 steht für false. • Bit 0 (LSB): Anwesenheit des Benutzers (UP) • Bit 1: Unbenutzt • Bit 2: Benutzer gegenüber Token authentifiziert (UV) • Bit 3-5: Unbenutzt • Bit 6: Authentifikatordaten enthalten attestedCredentialData (AT) • Bit 7 (MSB): Authentifikatordaten enthalten Erweiterungen des CTAP-Protokolls (ED)
signCount	32-bit unsigend Integer. Gibt die Anzahl der bereits vom Token generierten Signaturen an.
attestedCredentialData	Feld nur bei Registrierung vorhanden, enthält Daten zum Schlüsselpaar. Diese umfassen die AAGUID des Tokens, die Länge der *credentialID*, die *credentialID* selbst und den öffentlichen Schlüssel des zuvor generierten dienstspezifischen Schlüsselpaares.
extensions	Alle in der aktuellen Operation berücksichtigten Erweiterungen des Protokolls und die dazugehörigen berechneten Werte.

Tabelle 4: Darstellung der Datenstruktur der *Authentifikatordaten* nach [4].

Weiterhin wird ersichtlich, dass weder in den *clientData* noch in den *Authentifikatordaten* persönliche, die Identität des Benutzers betreffende, Informationen enthalten

sind. Selbst die erfolgreiche Authentifizierung des Benutzers sowie die Prüfung auf Anwesenheit eines Benutzers werden über je ein Flag an den Server übermittelt. Alle weiteren Daten sind ebenfalls losgelöst von der Identität des legitimen Benutzers und stellen Informationen zum Server oder allgemeine Informationen zum Authentifikator dar. Somit kann, sofern das Vertrauen zwischen Authentifikator und Server bereits etabliert wurde, die Authentifizierung gegenüber dem Server ausschließlich auf Grundlage der mit dem dienstpspezifischen privaten Schlüssel erstellten Signaturen erfolgen.

4.1.5 Sicherheitsbetrachtungen

Wie bereits für U2F und UAF in Kapitel 2.2.3 aufgeführt, gelten die in der FIDO Security Reference [57] erstellten Analysen von Schutzzielen, möglichen Angriffsvektoren und Gegenmaßnahmen ebenfalls für FIDO2. Neben den dort bereitgestellten Informationen sind jedoch einige WebAuthn-spezifische Sicherheitsbetrachtungen notwendig. Hierzu werden in der Spezifikation [4] sowohl die Vorteile von WebAuthn bezüglich der Sicherheit als auch mögliche Schwachstellen und damit verbundene Sicherheitsrisiken dargestellt.

Das in der Spezifikation [4] definierte Design der Schnittstellen und der Authentifikatoren bietet dabei einige Vorteile in der Benutzung und Sicherheit. Diese zeigen sich in einer einheitlichen Schnittstelle, die eine große Kompatibilität mit vielen Diensten ermöglicht. Dabei wird durch das Protokoll ein Schutz gegen Man-in-the-middle-Angriffe geboten. Weiterhin kann die Personalisierung lokal durch den Benutzer ausgeführt werden, da die biometrischen Daten nur auf dem Token selbst hinterlegt werden. Jeder Hersteller kann daher ein eigenes Tool zur Personalisierung des Tokens bereitstellen oder dieses in das Token selbst integrieren. Dadurch, dass die für die Benutzerauthentifizierung verwendeten Verfahren durch WebAuthn nicht unterschieden werden, kann so auf leichte Weise eine Multi-Faktor-Authentifizierung von jedem Dienst bereitgestellt werden. Da zudem jedes FIDO2-zertifizierte Token mit der WebAuthn-Schnittstelle funktioniert, kann der Benutzer aus einer großen Anzahl an anwendbaren Token auswählen. Hierbei kann der Nutzer ein seinen Ansprüchen entsprechenden Authentifikator wählen und diesen anschließend theoretisch für eine beliebige Anzahl von Diensten verwenden.

Jedoch können die Implementierungen verschiedene Schwachstellen aufweisen, sodass nicht mehr alle zuvor genannten Vorteile zutreffen [4]. Erfolgt beispielsweise die Generierung der Aufgabe, welche in den *clientDataHash* mit einfließt, nicht zufällig oder die Aufgabe kann leicht erraten werden oder ist statisch, so kann ein Replay-Angriff durch abgefangene Nachrichten erfolgen. Hierbei können die abgefangenen Antworten des Tokens durch den Angreifer während der Registrierung oder Authentifizierung gegenüber einem Dienst wieder eingespielt werden. Somit wäre der Schutz vor Man-in-the-middle-Angriffen nicht mehr gegeben. Daher sollten entsprechende Aufgaben mindestens 16 Byte lang sein und zufällig generiert werden.

Neben der Aufgabengenerierung birgt auch die Self Attestation Schwachstellen hinsichtlich Man-in-the-middle-Angriffen [4]. Da bei der Self Attestation die Signatur mit einem während der Registrierung selbst generierten Schlüsselpaar erstellt wird, kann

ein Angreifer die Nachricht an das Token sowie die entsprechende Antwort abfangen und anschließend ein eigenes Schlüsselpaar etablieren. Dazu muss der Angreifer aus den abgefangenen Nachrichten eine gefälschte Antwort erstellen, wobei für die Erstellung der Signatur ein vom Angreifer generiertes Schlüsselpaar verwendet wird. Anschließend sendet der Angreifer, anstelle der abgefangenen legitimen Antwort, die gefälschte Antwort an den Clienten zurück. Die gefälschte Antwort wird nun an den Dienst weitergegeben. Da das Vertrauen auf dem soeben generierten Schlüsselpaar liegt, kann hier kein Vertrauensbruch festgestellt werden. Das falsche Schlüsselpaar ist etabliert. Doch nicht nur die Self Attestation bietet diesen Angriffsvektor. Auch der Verzicht auf die Verwendung eines Vertrauensmodells oder auf die Validierung der Signatur durch den Server ermöglicht den zuvor beschriebenen Angriff. Jedoch würde der Benutzer die Etablierung des falschen Schlüsselpaares spätestens bei der nächsten Authentifizierung bemerken, da diese mit dem vom legitimen Benutzer verwendeten Token nicht möglich ist, da der private Schlüssel des Tokens nicht mit dem auf dem Server hinterlegten öffentlichen Schlüssel zusammenpasst.

Weitere Probleme können durch kompromittierte Zertifikate bei Verwendung der Basic Attestation oder Attestation CA entstehen [4]. Wird eine Zertifizierungsstelle eines Zertifikats der Zertifikatskette einer Geräteserie kompromittiert, hat dies keine Auswirkungen auf das dazugehörige Schlüsselpaar. Die Vertrauenswürdigkeit der Zertifikate ist zwar nicht mehr gegeben, der Hersteller kann jedoch durch andere Zertifizierungsstellen neue Zertifikate für das bereits hinterlegte Schlüsselpaar ausstellen lassen. Daraufhin können die Tokens wie gewohnt verwendet werden. Wird jedoch das auf dem Token hinterlegte Zertifikat kompromittiert, muss ein neues Schlüsselpaar mit dazugehörigem Zertifikat für die entsprechende Geräteserie mittels eines Firmwareupdates bereitgestellt werden, um die Geräteserie weiterhin für die Registrierung bei neuen Diensten nutzen zu können. Das Kompromittieren des Schlüsselpaares ermöglicht dabei dem Angreifer das Erstellen legitimer Signaturen während der Registrierung des Tokens, wodurch ein Man-in-the-middle-Angriff, wie zuvor beschrieben, durchgeführt werden kann. Ist ein Update nicht möglich oder wurde das Update vom Benutzer nicht durchgeführt, kann somit den generierten Signaturen der Geräteserie nicht mehr vertraut werden und entsprechende Registrierungsversuche sollten durch die Dienste abgelehnt werden. Da dieses Schlüsselpaar nur die Etablierung des Vertrauens zwischen Token und Server betrifft, jedoch nicht die dienstspezifischen Schlüsselpaare, können bereits registrierte Dienste weiterhin mit dem Token verwendet werden. Ein weiteres Problem für die Attestation CA stellt zudem die verwendete Zertifizierungsstelle dar. Ist diese nicht mehr verfügbar, kann das Token keine neuen Schlüsselpaare und die dazugehörigen Zertifikate anfordern. Somit ist eine Benutzung des Tokens nicht mehr möglich.

Auch der Verlust eines Tokens stellt den Benutzer vor Probleme, da die Authentifizierung gegenüber den Diensten, bei denen das Token hinterlegt wurde, nicht mehr möglich ist [4]. Dies ist darauf zurückzuführen, dass die etablierten Schlüsselpaare in der Regel an das Token gebunden sind und somit nicht mehr auf die benötigten privaten Schlüssel zugegriffen werden kann. Um dieses Problem zu verhindern, ermöglicht WebAuthn die Registrierung mehrerer verschiedener Tokens für einen Dienst. Da die Tokens für die passwortlose und benutzerdatenlose Authentifizierung zudem an den Benutzer

gebunden werden, stellt der Verlust zwar ein Sicherheitsrisiko dar, allerdings kann der mögliche Angreifer die Authentifizierung gegenüber dem Token, in der Regel, nicht ohne zusätzlichen Aufwand erfolgreich abschließen.

4.1.6 Privatsphäre

Neben der Sicherheit des Protokolls ist auch die Wahrung der Privatsphäre der einzelnen Benutzer wichtig. Dazu wurden durch die FIDO Alliance Prinzipien hinsichtlich des Schutzes der Privatsphäre in den FIDO Privacy Principles [64] festgehalten sowie die Privatsphäre im WebAuthn-Standard [4] betrachtet. Dies stellt jedoch eine Herausforderung dar, weil das Protokoll zum einen gewährleisten muss, dass nur der legitime Benutzer Zugriff zu den Diensten erhalten kann. Zum anderen darf den Diensten jedoch keine Möglichkeit gegeben werden, den Benutzer auch über die Grenzen des Dienstes hinweg zu identifizieren und das Verhalten des Nutzers zu analysieren.

Zu diesem Zweck hat sich die FIDO Alliance zum Ziel gesetzt, Protokolle bereitzustellen, die so wenig persönliche Daten des Benutzers sammeln und austauschen wie möglich [64]. Dazu werden nur die notwendigsten Daten für die Identifizierung erhoben. Sensible Daten, wie z.B. biometrische Samples, werden nur lokal gespeichert. Zudem wird darauf geachtet, dass der Kontext jeder mit dem Authentifikator ausgeführten Operation dem Benutzer ersichtlich ist und diese Operation durch den Benutzer bestätigt wurde. Auch der Zugriff auf sensible Daten soll so geschützt werden, dass eine Identifizierung des Benutzers während der Registrierung bzw. Authentifizierung bei zwei oder mehreren verschiedenen Diensten nicht ermöglicht wird. Der Benutzer soll weiterhin auch die Möglichkeit haben, die bei einem Dienst verwendeten Authentifikatoren und die damit verbundenen Daten verwalten zu können.

Um die Identifizierung des legitimen Benutzers gegenüber einem Dienst gewährleisten zu können, müssen jedoch verschiedene persönliche Daten über die dahinterstehende Person erhoben werden [4]. Diese Daten sowie weitere den Authentifikator identifizierende Daten ermöglichen die potentielle Herstellung von Verbindungen zwischen mehreren Operationen und somit die potentielle Offenlegung der betreffenden Person oder die Analyse des Benutzerverhaltens. Zu diesen Daten zählen beispielsweise Benutzernamen, E-Mail-Adressen und biometrische Daten. Auch die Erhebung der Geräteserie sowie der Seriennummer des Authentifikators oder die erzeugte *credentialID* zur Validierung der erzeugten Signaturen sind gegebenenfalls notwendig. Für diese Daten muss somit ein entsprechender Schutz gewährleistet werden, sodass eine Deanonymisierung nur schwer möglich ist.

Zu diesem Zweck wird der Schutz der Privatsphäre in das Design des Protokolls einbezogen [4]. *CredentialIDs* wurden daher so definiert, dass diese nur das für die Authentifizierung verwendete Schlüsselpaar identifizieren und keine Benutzerdaten enthalten. Zudem werden die Schlüsselpaare an den Dienst, für den die Schlüssel erstellt wurden, gebunden, sodass nur der entsprechende Dienst den Zugriff des Tokens auf den privaten Schlüssel des Schlüsselpaares veranlassen kann. Bei der Generierung der *credentialID* wird auch darauf geachtet, dass zwei verschiedene *credentialIDs* eines Tokens nicht miteinander in Verbindung gebracht werden können. Hierzu ist beispielsweise

die Vergabe einer zufällig generierten ID für eine *credentialID* vorgesehen. Dennoch kann die *credentialID* während der Registrierung oder der passwortlosen Anmeldung bei einem Dienst mit dem Benutzernamen oder der E-Mail-Adresse des Benutzers in Verbindung gebracht werden. Um dies zu umgehen, können Dienste eine benutzerdatenlose Registrierung und Anmeldung mit FIDO2-Tokens anbieten. Dabei wird eine entsprechende BenutzerID vom Server des Dienstes generiert und vom Token in die Generierung der *credentialID* mit einbezogen. Anhand dieser BenutzerID kann der Server anschließend während des Authentifizierungsprozesses gegenüber dem Dienst den Nutzer intern identifizieren und die Anmeldung durchführen.

Jedoch können die für die beiden Vertrauensmodelle *Basic Attestation* und *Attestation CA* verwendeten Zertifikate ein Risiko für die Privatsphäre des Benutzers darstellen [4]. Werden bei der *Basic Attestation* nur wenige Geräte mit dem gleichen Zertifikat in Umlauf gebracht, können über das Zertifikat schnell Rückschlüsse – auch dienstübergreifend – auf die Nutzer gezogen werden. Um diese Gefahr zu verringern wird daher empfohlen, eine entsprechend große Anzahl an Geräten derselben Serie zu produzieren und an eine große Benutzergruppe zu verteilen. Auf diese Weise wird es den Diensten erschwert, einen spezifischen Nutzer anhand des Zertifikats zurückzuverfolgen. Die gleiche Problematik tritt bei Wiederverwendung der für die *Attestation CA* generierten Zertifikate auf. Da diese dynamisch generiert werden, ist jedes Zertifikat eindeutig durch verschiedene Dienste identifizierbar. Damit dies verhindert wird, sollte für jede Registrierung des Schlüssels gegenüber eines Dienstes ein neues Schlüsselpaar und ein dazugehöriges Zertifikat ausgestellt werden. Gleiches gilt für die generierten Schlüsselpaare bei Anwendung der ECDAA.

Weiterhin sollte jede Operation sowie die Herausgabe von Daten nur dann erfolgen, wenn die Authentifizierung des Benutzers gegenüber dem Token erfolgreich durchgeführt wurde [4]. Auf diese Weise wird die Erhebung von FIDO- und benutzerspezifischen Daten, wie z.B. die Ermittlung der mit dem Token verwendeten Dienste, durch Dritte verhindert. Eine Erhebung könnte sonst erfolgen, indem z.B. testweise unaufgefordert Anfragen an einzelne Tokens gesendet und anschließend deren Antworten ausgewertet werden. Wird zudem ohne das Wissen des Benutzers eine Operation ausgelöst, sollte diese nicht sofort abgebrochen oder ohne Zustimmung des Benutzers bestätigt werden, sondern ein vom Dienst definiertes Timeout zum Abbruch der Operation führen. Auf diese Weise werden keine Informationen über eventuell nicht angeschlossene Tokens oder nicht vorhandene *credentialIDs* herausgegeben. Zudem wird ein Denial-of-Service-Angriff durch die ungewollte Ausführung des Werksresets damit verhindert. Sollten sich mehrere Benutzer ein Token teilen, muss außerdem gewährleistet werden, dass die Daten der jeweiligen Benutzer voneinander getrennt werden und jeder Benutzer nur über seine eigenen Daten und privaten Schlüssel verfügen kann.

4.2 Client to Authenticator Protocol

Die Spezifikation des Client to Authenticator Protocols (CTAP) [5] beschreibt die Kommunikation zwischen Clienten und Authentifikatoren. Dazu wird eine Schnittstelle für Authentifikatoren definiert, welche diese implementieren müssen. Die CTAP-

Spezifikation definiert dabei zwei Versionen des Protokolls, zum einen CTAP1 / U2F für die Abwärtskompatibilität zu U2F-Tokens sowie zum anderen CTAP2 zur Verwendung von FIDO2-Tokens. Weiterhin wird auf die verschiedenen Transportverfahren USB, NFC und Bluetooth eingegangen und es werden entsprechende Nachrichtenformate definiert. Dabei werden die Nachrichten für den Transport in der Concise Binary Object Representation (CBOR) [9] kodiert.

Zu Beginn dieses Abschnitts werden die Protokolle CTAP2 und CTAP1 / U2F vorgestellt und daraufhin das Datenformat CBOR betrachtet. Abschließend werden die Transportverfahren beschrieben.

4.2.1 CTAP2

CTAP2 ist die zweite Version des CTAP-Protokolls [5] und definiert eine Schnittstelle zur Verwendung von FIDO2-Authentifikatoren. Die einzelnen Operationen der Schnittstelle können dabei unabhängig voneinander und asynchron ausgeführt werden. Das Protokoll garantiert jedoch keine Ordnung oder Zuverlässigkeit im Nachrichtenaustausch. Sind diese Eigenschaften notwendig, müssen diese durch das Transportprotokoll oder die Anwendung selbst gewährleistet werden.

Die Schnittstelle definiert Methoden und Datenstrukturen, welche durch die Authentifikatoren bereitgestellt werden müssen. Nachfolgend werden die einzelnen Methoden vorgestellt und die notwendigen Ein- und Ausgaben beschrieben.

authenticatorMakeCredential() wird während des Registrierungsprozesses zur Generierung eines neuen dienstspezifischen Schlüsselpaares und zur Etablierung des Vertrauens verwendet. Dazu werden der *clientDataHash*, die Serverinformationen, einige Benutzerinformationen sowie eine Liste der vom Server unterstützten Algorithmen für die Generierung des Schlüsselpaares benötigt. Weitere Angaben wie z.B. Erweiterungen des Protokolls oder zu den notwendigen Eigenschaften des Authentifikators sind dabei optional. Wird die Operation erfolgreich durchgeführt, wird anschließend das *attestationObjekt* zurückgegeben. Dieses umfasst neben den *Authentifikatordaten* und der *credentialID* auch die Attestation, anhand derer ein Server das Vertrauen in das Token und die generierte *credentialID* beurteilen kann.

authenticatorGetAssertion() wird während des Anmeldevorgangs oder einer Transaktionsbestätigung zur Erstellung der Signatur mittels des zuvor etablierten dienstspezifischen Schlüsselpaares verwendet. Um diese Operation ausführen zu können, werden die *ServerID* und *der clientDataHash* benötigt. Anhand dieser Informationen kann der Authentifikator den zuvor generierten privaten Schlüssel des etablierten Schlüsselpaares suchen und die Signatur über den *clientDataHash* und die *Authentifikatordaten* erstellen.

Je nach Eigenschaften des Tokens oder der Anzahl der für das Benutzerkonto hinterlegten Tokens, wird jedoch eine *allowList* benötigt, welche ein oder mehrere *credentialIDs* auflistet. Alle in der Liste enthaltenen *credentialIDs* können

dabei potentiell zur Authentifizierung des Benutzers verwendet werden. Wird eine solche Liste übermittelt, wird durch **authenticatorGetAssertion()** nur das erste Element der Liste entnommen und zur Generierung der Signatur verwendet. Alle weiteren Elemente werden durch **authenticatorGetNextAssertion()** behandelt.

Wurde die Signatur erstellt, werden unter anderem die *Authentifikatordaten*, die *Signatur* und ggf. die verwendete *credentialID* sowie die ausstehende Anzahl an Elementen der Liste zurückgegeben. Kann die *credentialID* aus der Liste nicht korrekt gelesen werden, wird dies als fehlender Parameter für den Funktionsaufruf als Fehler behandelt.

authenticatorGetNextAssertion() wird während des Registrierungsprozesses aufgerufen, wenn mehr als eine *credentialID* per *allowList* mittels **authenticatorGetAssertion()** übermittelt wurde. Dieser Befehl folgt somit immer auf **authenticatorGetAssertion()** und nimmt daher keine Parameter entgegen. Wird **authenticatorGetNextAssertion()** aufgerufen, wird die nächste *credentialID* aus der Liste entnommen und eine Signatur über den bereits übermittelten *clientDataHash* sowie die *Authentifikatordaten* berechnet. Anschließend werden, bis auf die Anzahl der verbleibenden Elemente der Liste, dieselben Daten wie bei **authenticatorGetAssertion()** zurückgegeben. Anstelle der Angabe über die Anzahl verbleibender Elemente, wird nun die Anzahl erstellter Signaturen zurückgegeben. Sobald der Zähler gleich oder größer der Anzahl der Elemente der Liste ist, wird ein Fehler zurückgegeben.

Der Aufruf von **authenticatorGetNextAssertion()** muss für jede *credentialID* der Liste separat erfolgen. Ist eine *credentialID* nicht bekannt, verhält sich die Funktion wie **authenticatorGetAssertion()**.

authenticatorGetClientPin() wird zur Verwendung des *clientPin* benötigt. Wird dieses Verfahren unterstützt, kann bei erstmaliger Anwendung des Tokens über den Clienten eine PIN vergeben werden, welche auf dem Token etabliert wird. Diese PIN wird anschließend bei jeder auszuführenden Operation zur Benutzerauthentifizierung gegenüber dem Token abgefragt. Die dazu verwendete PIN wird dabei nur während der erstmaligen Personalisierung oder während der Änderung vollständig, aber verschlüsselt übertragen. Ansonsten wird bei jedem neuen Energiezyklus des Authentifikators ein *pinToken* etabliert, welches für die weiteren Operationen verwendet wird.

Damit alle Funktionalitäten ermöglicht werden können, definiert **authenticatorGetClientPin()** sieben weitere Unterbefehle. Je nach Bedarf wählt der Client den entsprechenden Unterbefehl, um die gewünschte Operation durchzuführen. Nachfolgend werden die Unterbefehle aufgelistet.

- getRetries() – Über diesen Unterbefehl können die ausstehenden Versuche zur Benutzerauthentifizierung abgefragt werden. Wurde die PIN zu oft

hintereinander falsch eingegeben und es stehen keine erneuten Versuche aus, wird das Token blockiert.

- getKeyAgreement() – Diese Operation wird dazu verwendet, ein geteiltes Geheimnis zwischen Client und Authentifikator zu etablieren. Dieses Geheimnis wird für die Verschlüsselung der PIN und davon abgeleiteter Werte benötigt.
- setPin() – Etabliert den bei der Personalisierung gewählten PIN auf dem Token.
- changePin() – Ändert die aktuell auf dem Token etablierte PIN zu einer neuen, vom Benutzer gewählten, PIN.
- getPINToken() – Übermittelt nach korrekter PIN-Authentifizierung das für den aktuellen Energiezyklus generierte *pinToken* an den Clienten. Dieses wird daraufhin für alle weiteren Operationen desselben Energiezyklus anstelle des PINs verwendet.

Attribut	Bedeutung	Beschreibung
plat	platform device	Gibt an, ob das Token in das Gerät fest integriert ist (true) oder nicht (false).
rk	resident key	Gibt an, ob die privaten Schlüssel des Schlüsselpaares auf dem Gerät hinterlegt werden können (true) oder nicht (false).
clientPin	clientPin	Gibt an, ob das Gerät in der Lage ist, den *clientPin* zu benutzen, um den User zu authentifizieren. Wird das Attribut weggelassen, ist die Möglichkeit nicht gegeben. Ist es auf false gesetzt, muss der *clientPin* zuerst initialisiert werden und kann daraufhin verwendet werden. Hat das Attribut den Wert true, so kann der *clientPin* normal verwendet werden.
up	user presence	Gibt an, ob das Gerät die Anwesenheit des Benutzers überprüfen kann.
uv	user verification	Gibt an, ob das Gerät den Benutzer authentifizieren kann – dies kann z.B. durch verbaute biometrische Sensoren geschehen. Wird das Attribut weggelassen, ist die Möglichkeit nicht gegeben. Ist es auf false gesetzt, muss das Gerät zuerst durch den Benutzer personalisiert werden und kann daraufhin verwendet werden. Hat das Attribut den Wert true, so kann der Authentifikator normal verwendet werden.

Tabelle 5: Mögliche Attribute eines Authentifikators und deren Bedeutung nach [5].

authenticatorGetInfo() kann von dem Clienten bzw. Dienst dazu verwendet werden, die Eigenschaften und Möglichkeiten des Authentifikators zu ermitteln. Dies

sind Informationen über die unterstützte CTAP-Version, die AAGUID des Authentifikators, die maximale unterstützte Nachrichtenlänge, die Attribute des Authentifikators sowie das ggf. verwendete PIN-Protokoll für *clientPin*. Tabelle 5 gibt dabei eine mögliche Übersicht über alle Attribute des Authentifikators. Die Attribute werden dabei als Wahrheitswerte in der Nachricht kodiert.

authenticatorReset() wird für das Zurücksetzten des Authentifikators auf Werkseinstellungen verwendet. Da diese Funktion für jede Geräteserie spezifisch implementiert werden muss, werden hier keine Vorgaben zur Implementierung gegeben. Das Ergebnis der Funktion muss jedoch der Werksreset und die damit verbundene Invalidierung aller mit dem Token etablierten Schlüsselpaare sein.

Nachfolgend wird die Kommunikation zwischen Client und Token der zuvor beschriebenen Schnittstelle anhand zweier Beispiele, basierend auf [5, 4], beschrieben. Die dort dargestellten Abläufe können – je nach Client und Token – variieren. Für diese Beispiele wird davon ausgegangen, dass ein biometriebasiertes Token ohne internen Speicher benutzt wird, welches *Basic Attestation* zur Etablierung des Vertrauens verwendet.

Das erste Beispiel wird in Abbildung 6 dargestellt und verdeutlicht die Registrierung des Tokens bei einem Dienst. Hierzu übermittelt der Server die benötigten Daten, welche auch Informationen zum Benutzer beinhalten. Diese Benutzerdaten umfassen, wie in [4] beschrieben, mindestens eine dienstspezifische AccountID zur Identifizierung des Benutzers durch den Dienst. Diese wird benötigt, falls das Token benutzerdatenlose Authentifizierung unterstützt. Weiterhin können der Benutzername, der Anzeigename des Accounts sowie eine URL zur Darstellung eines Icons übermittelt werden. Die optionalen Daten sind jedoch nur zur Anzeige durch den Clienten bestimmt, damit der Benutzer nachvollziehen kann, für welchen Kontext seine Zustimmung benötigt wird. Anschließend kann durch den Clienten über den in Tabelle 3 (siehe S. 39) genannten Daten der *clientDataHash* berechnet werden. Daraufhin wird der Benutzer aufgefordert, das zu verwendende Token anzuschließen. Ist dies erfolgt, fragt der Client zuerst die Eigenschaften und Möglichkeiten des Authentifikators über authenticatorGetInfo() ab, um die Nachrichtenlänge und weitere Angaben zu bestimmen. Ist die Selbstauskunft durch den Authentifikator übermittelt worden, kann nun die Generierung des Schlüsselpaares und der Signatur durch den Aufruf von authenticatorMakeCredential(), mit allen vom Token benötigten Daten, angefragt werden. Bevor die Operation durch das Token ausgeführt wird, muss der Benutzer seine Zustimmung geben, indem er sich über die biometrische Authentifizierung gegenüber dem Token verifiziert. Anschließend können die Generierung des dienstspezifischen Schlüsselpaares sowie der Signatur mit dem geräteseriespezifischen privaten Schlüssel des Tokens über den in Tabelle 4 (siehe S. 40) dargestellten *Authentifikatordaten* und dem *clientDataHash* erfolgen. Daraufhin werden die generierten Daten an den Clienten geschickt, welcher diese um die *clientData* erweitert und an den Server sendet. Abschließend validiert der Server die erhaltenen Daten und ermittelt das Vertrauen in den Authentifikator. Bei erfolgreicher Validierung ist das Schlüsselpaar etabliert.

Der in Abbildung 7 dargestellte Ablauf einer Authentifizierung gegenüber eines Dienstes stellt das zweite Beispiel dar und gleicht dem zuvor beschrieben Vorgang zur

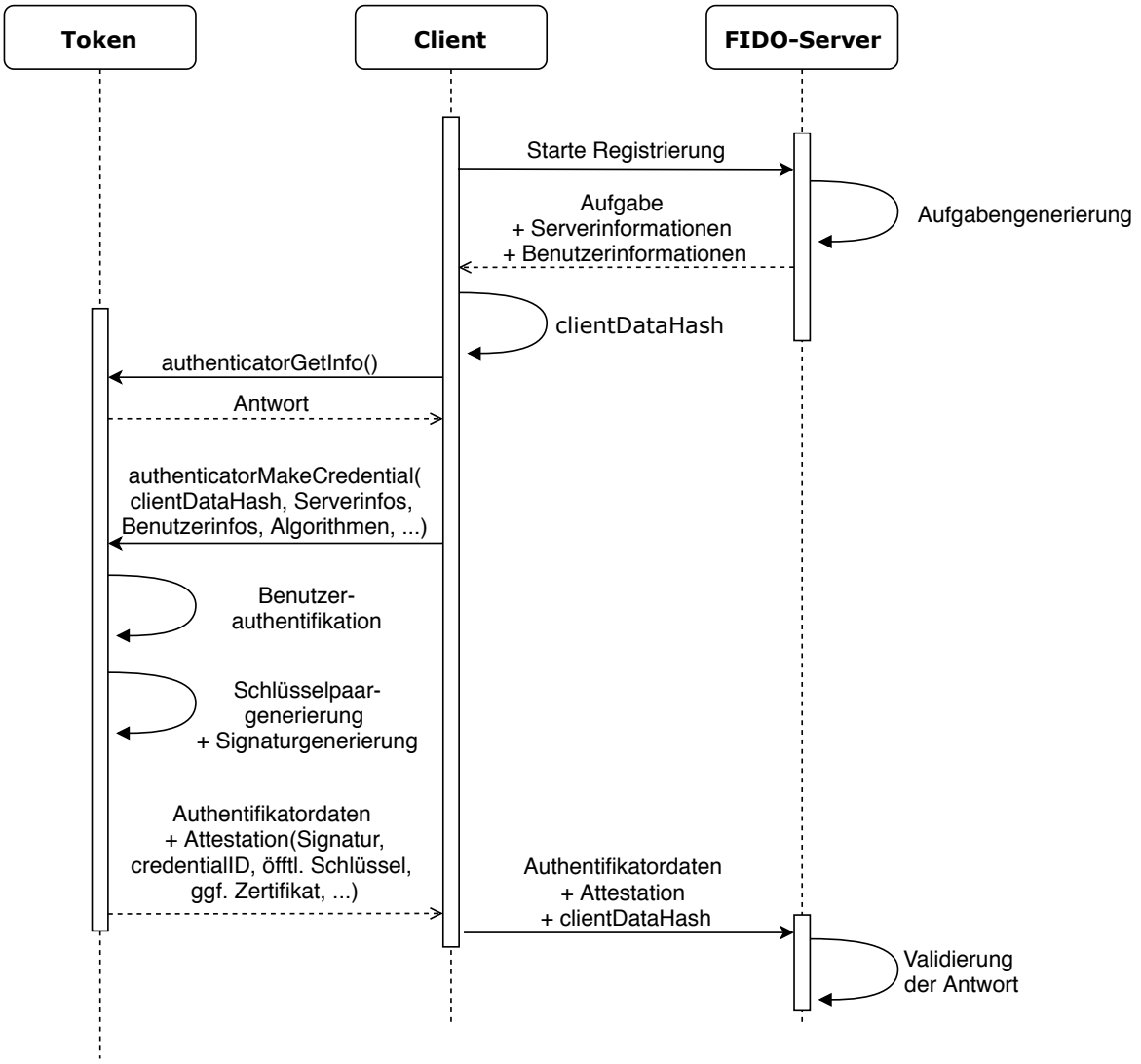

Abbildung 6: Beispielhafter Ablauf einer authenticatorMakeCredential()-Operation zur Registrierung des Tokens unter Verwendung eines biometriebasierten Authentifikators. Angelehnt an [5, 4].

Registrierung in vielen Punkten. Jedoch werden im direkten Vergleich zwei Unterschiede ersichtlich. Es müssen weniger Daten für die Authentifizierung zwischen Server, Client und Token ausgetauscht werden, da die Bindung an den Kontext und die Ermittlung des Vertrauens bereits erfolgt ist. Der Beweis, dass der korrekte dienstspezifische private Schlüssel zur Erstellung der Signatur verwendet wurde, reicht nun aus, da nur der ausstellende Authentifikator auf den passenden privaten Schlüssel zugreifen konnte. Weiterhin muss der Server anstelle der *Benutzer-* und *Serverinformationen* nur noch die für den Account hinterlegte *credentialID* übermitteln. Dies ist darauf zurückzuführen, dass das Token keinen internen Speicher besitzt und daher die Speicherung des privaten Schlüssels auf den Server ausgelagert werden musste.

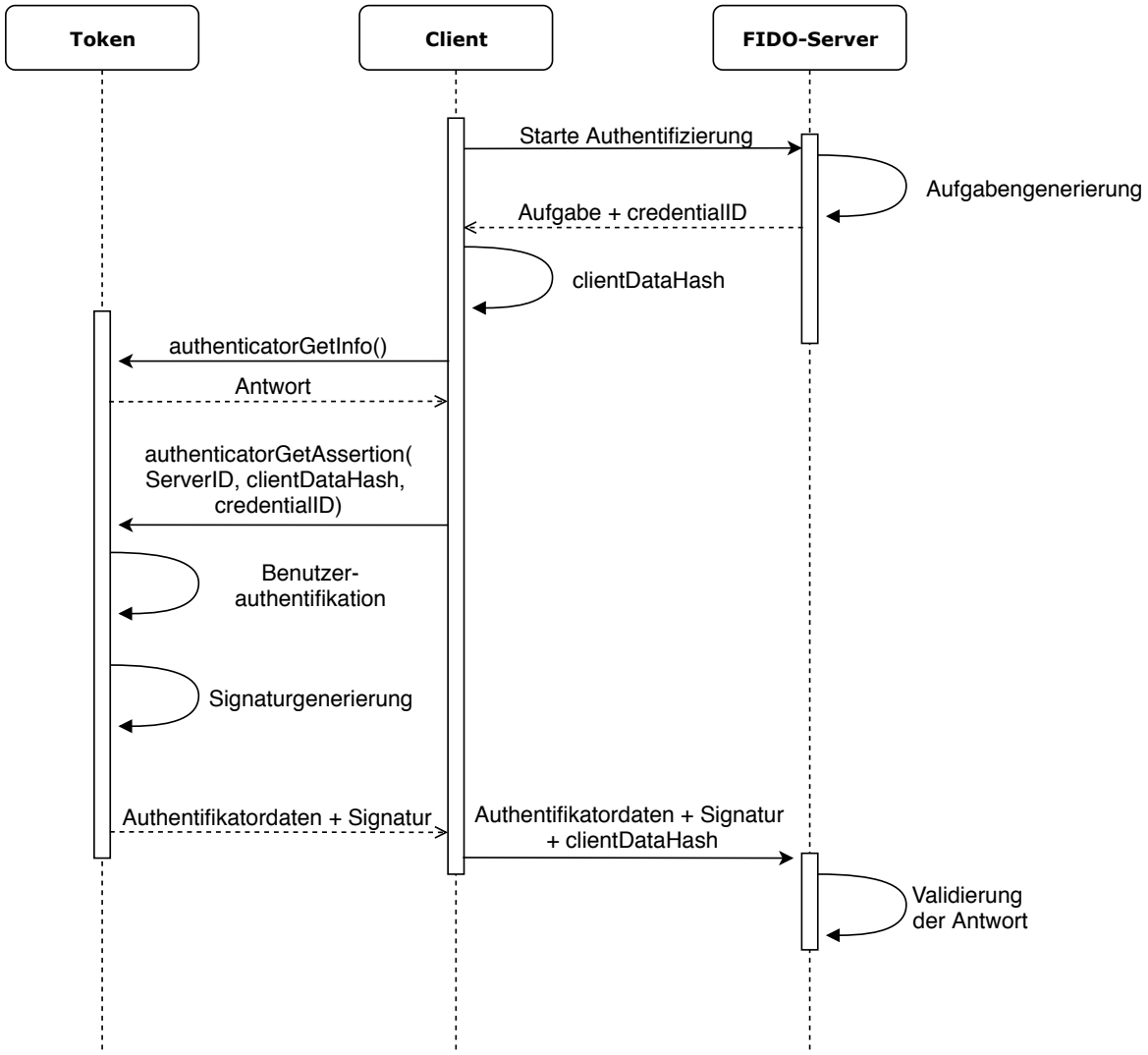

Abbildung 7: Beispielhafter Ablauf einer authenticatorGetAssertion()-Operation zur Authentifizierung gegenüber des Dienstes unter Verwendung eines biometriebasierten Tokens. Angelehnt an [5, 4].

4.2.2 CTAP1 / U2F

CTAP1 / U2F stellt die erste Version des Client to Authenticator Protocol [5] und die Abwärtskompatiblitität der CTAP2-Schnittstelle zum 2FA-Standard U2F [17] dar. Diese Abwärtskompatibilität ermöglicht dadurch die Anwendung von bereits erworbenen und weiterhin angebotenen U2F-Tokens mit WebAuthn. Um diese zu ermöglichen, wird eine Übersetzung von CTAP2-Anfragen auf CTAP1 / U2F-Anfragen und CTAP1 / U2F-Antworten auf CTAP2-Antworten definiert.

Die Übersetzung wird dabei vom Clienten durchgeführt. Dabei überprüft der Client alle ausgehenden Anfragen auf die enthaltenen Parameter. Werden Erweiterungen oder Eigenschaften gefordert, die durch U2F-Tokens nicht erfüllt werden können, wird

die Anfrage durch den Clienten abgebrochen. Ansonsten generiert der Client aus den vorliegenden Informationen eine U2F-kompatible Anfrage und sendet diese an das U2F-Token. Das Token generiert daraufhin eine Antwort und sendet diese an den Clienten. Die Antwort entspricht dem U2F-Nachrichtenformat, weshalb diese Antwort nun vom Clienten in eine CTAP2-kompatible Nachricht übersetzt werden muss. Da die für U2F generierte Antwort alle zwingend benötigten Informationen für CTAP2 enthält, kann diese Übersetzung durch den Clienten durchgeführt werden. Daten, die nicht enthalten sind – wie z. B. der *signCount*, die *AAGUID* oder die *Flags* – können dabei durch den Clienten mit Standardwerten belegt werden. Das heißt für den *signCount* und die *AAGUID* die Belegung mit Nullen. Bei den Flags hingegen wird das *UP*-Flag zum Signalisieren der Anwesenheit des Benutzers auf 1 gesetzt. Je nach ausgeführter Operation, wird ebenfalls das *AT*-Flag zur Angabe vorhandener *attestedCredentialData*, welche während der Registrierung verwendet werden, gesetzt. Alle weiteren Bits werden standardmäßig mit 0 belegt.

4.2.3 Concise Binary Object Representation

Die Concise Binary Object Representation (CBOR) [9] ist eine binäre Datenrepräsentation zur Kodierung von Nachrichten und wurde basierend auf der JavaScript Object Notation (JSON) [65] entwickelt. Ziel war es, eine möglichst kompakte und leicht de- und enkodierbare Darstellung für in Internetstandards verwendete Datenformate bereitzustellen. Dies soll es auch Geräten mit geringen Ressourcen und eingeschränkten Befehlssätzen ermöglichen, diese Formatierung zu verwenden. Weiterhin wurde auf die Kompatibilität zu JSON geachtet, sowie die Erweiterbarkeit der Datenrepräsentation bedacht.

Der Standard [9] definiert dabei acht Haupttypen zur Kodierung von Werten – unsigned Integer (Typ 0), negative Integer (1), Bytestrings (2), Textstrings (3), Arrays (4), Maps (5), Tags (6) und simple Datentypen (7) wie z.B. Wahrheitswerte. Neben den inhaltslosen Datentypen wie Wahrheitswerte umfassen die simplen Datentypen jedoch auch die Kodierung von Fließkommazahlen. Dabei folgt die Kodierung stets dem gleichen Muster. Das erste Byte der Kodierung eines Wertes beschreibt in den ersten 3 Bits den Datentyp, die darauffolgenden 5 Bits kodieren zusätzliche Informationen. Diese zusätzlichen Informationen kodieren dabei die Werte selbst oder die Länge des zu kodierenden Wertes in Bytes. Für Maps und Arrays bedeutet die Länge die Anzahl der Paare bzw. der Elemente der Datenstruktur. Hierbei ist der Wert zur direkten Kodierung der Werte bzw. Länge mit 23 nach oben beschränkt. Um größere Werte zu kodieren kann innerhalb der 5 Bits für zusätzliche Informationen der Wert 24, 25, 26 oder 27 hinterlegt werden. Diese Werte geben anschließend an, wie viele darauffolgende Bytes für die Kodierung des Wertes oder der tatsächlichen Länge benötigt werden. Dies kann entweder eine 1-, 2-, 4- oder 8-byte unsigned Integer sein. Tabelle 6 zeigt einige beispielhafte Kodierungen von Werten. Daraus geht hervor, dass ein Byte zur Kodierung von kleinen Zahlen oder Wahrheitswerten ausreicht. Zudem wird ersichtlich, dass die Länge der CBOR-Kodierung im ungünstigsten Fall der Länge einer vergleichbaren

		Kodierung		
Wert	Typ	1. Byte	Länge o. Wert	Wert
7	0	b000_00111		
24	0	b000_11000	00011000	
Hallo Welt!	3	b011_01010		48 61 6c 6c 6f 20 57 65 6c 74
Hallo Welt in mehr als 23 Zeichen!	3	b011_11000	00100010	48 61 6c 6c 6f 20 57 65 6c 74 20 69 6e 20 6d 65 68 72 20 61 6c 73 20 32 33 20 5a 65 69 63 68 65 6e 21
[7, 7, 25]	4	b100_00011		07 07 24 25
True	7	b111_10101		

Tabelle 6: Beispielhafte CBOR-Kodierungen von Werten der Datentypen unsigned Integer (Typ 0), Textstring (Typ 3), Array (Typ 4) von unsigned Integern (Typ 0) sowie simpler Datentyp (Typ 7) nach [9].

Tag-Length-Value-Kodierung (TLV) entspricht, welche ein Byte für den Typ, acht Bytes für die Länge des Wertes sowie den Wert selbst umfasst.

Die CTAP-Protokolle [5] legen eine in der CBOR-Spezifikation [9] beschriebene selbst definierbare kanonische Form der Repräsentation fest. Bei dieser Definition werden alle Datentypen, außer Tags (Typ 6) unterstützt. Werte müssen zudem in der kanonischen Form der FIDO Alliance eine definite Länge zum Zeitpunkt der Kodierung aufweisen, da bei der normalen CBOR-Kodierung auch Werte unbestimmter Länge übertragen werden können. Die Länge eines Wertes sowie von Integers muss zudem so kompakt wie möglich kodiert werden. Weiterhin müssen die in Maps enthaltenen Paare anhand ihrer CBOR-kodierten Schlüssel lexikographisch sortiert werden. Die minimale Nachrichtenlänge, die in der CBOR-Kodierung von einem Authentifikator unterstützt werden muss, beträgt 1024 Bytes und darf eine Schachtelung bei Maps und Arrays von max. vier Stufen aufweisen.

4.2.4 Transportprotokolle

Zur Kommunikation zwischen Client und Token werden drei verschiedene Transportprotokolle in CTAP [5] definiert. Authentifikatoren und Clienten können dabei aus der Kommunikation via USB, NFC oder Bluetooth Low Energy wählen. Auch eine Unterstützung mehrerer Protokolle ist möglich. Dabei haben alle Protokolle gemein,

dass durch CTAP eigene proprietäre Nachrichtenformate definiert wurden. Die zu übertragenden Daten werden in CBOR kodiert und in Pakten mit einer Mindestlänge von 1024 Bytes übertragen. Im Folgenden wird kurz auf die einzelnen Transportprotokolle eingegangen.

USB Der Transport mittels USB wird über das Human Interface Device (HID) Protokoll definiert [5]. Dies ermöglicht die treiberlose Nutzung von Tokens an vielen Systemen sowie eine gleichzeitige Benutzung des Tokens durch viele verschiedene Anwendungen. Aufgrund der Gleichzeitigkeit muss jedoch eine transaktionale und atomare Ausführung der Operationen durch das Token gewährleistet werden. Eine Transaktion umfasst dabei die Anfrage eines Clienten und die dazugehörige Antwort des Tokens. Dies erfordert das Management des Kommunikationskanals sowie der Transaktionen, welches durch CTAP ebenfalls beschrieben wird.

NFC Im Gegensatz zu USB ermöglicht NFC [5, 66] die drahtlose Verwendung von Tokens gegenüber NFC-fähigen Endgeräten, wie z.B. Smartphones. NFC-Lesegeräte können darüber hinaus per USB an verschiedenen Plattformen eingerichtet werden, dafür ist jedoch zusätzliche Hardware notwendig. Aufgrund der Zeitbeschränkung einiger Plattformen zur Bearbeitung einer per NFC ausgelösten Operation, sollte die Antwort des Tokens innerhalb von $800ms$ erfolgen. Trotz drahtloser Verbindung bietet NFC den Vorteil, dass keine Batterie zur Stromversorgung benötigt wird, sondern diese durch das Lesegerät mittels elektromagnetischen Feldern über die Antenne induziert werden kann. Die maximale Entfernung für eine ausreichende Stromversorgung ist dabei – je nach Feldstärke des Lesegeräts – auf wenige Zentimeter beschränkt.

Bluetooth Low Energy Bluetooth Low Energy ist ein Protokoll, welches die drahtlose Verwendung mit Bluetooth-fähigen Endgeräten ermöglicht [5]. Anders als bei NFC, können hier theoretisch größere Distanzen überwunden werden. Für die Verwendung von Bluetooth LE wird eine Batterie zur Stromversorgung der Antenne und des Authentifikators benötigt. Um die Kommunikation zwischen Token und Endgerät herzustellen, müssen die beiden Geräte zudem in einem ersten Schritt gekoppelt werden. Nach der erfolgreichen Kopplung beider Geräte, sollte die weitere Kommunikation über Bluetooth LE darüber hinaus verschlüsselt werden, um Angriffe auf die drahtlose Verbindung zu unterbinden.

5 Zertifizierung

Die Zertifizierungen der FIDO Alliance dienen den Herstellern als Mittel zur Auszeichnung der Sicherheits- und Kompatibilitätseigenschaften ihrer Produkte. Insgesamt gibt es drei Zertifizierungskategorien: die funktionale Zertifizierung [19], die Zertifizierung von Authentifikatoren [6] und die Zertifizierung von biometrischen Komponenten [20]. Authentifikatoren können dabei in sechs verschiedene Sicherheitslevel eingestuft werden sowie für verschiedene FIDO Standards zertifiziert werden. Die Zertifizierung erfolgt

dabei nach von der FIDO Alliance festgelegten Kriterien, welche sich je nach Zertifizierung, Level und Standard unterscheiden. Neben der Bestätigung der Eigenschaften für den Hersteller bietet eine erfolgreiche Zertifizierung den Benutzern die Sicherheit, ein kompatibles und sicheres FIDO-Token zu erkennen und diese Tokens nach den eigenen Ansprüchen miteinander zu vergleichen.

Im Folgenden wird auf den Zertifizierungsprozess eines Authentifikators eingegangen. Abschließend werden die Zertifizierungslevel für Authentifikatoren kurz vorgestellt.

5.1 Zertifizierungsprozess

Die in [6, 19, 20] beschriebenen Zertifizierungsrichtlinen der FIDO Alliance umfassen alle Informationen zu dem jeweiligen Zertifizierungsprozess. Damit die Zertifizierung eines Authentifikators [6] auf eines der Sicherheitslevel erfolgen kann, muss der Authentifikator zuvor die funktionale Zertifizierung [19] durchlaufen und bestehen. Verwendet der Authentifikator zudem biometrische Sensoren zur Authentifizierung des Benutzers und soll für Level 3 oder Level 3+ eingestuft werden, muss die verwendete biometrische Komponente ebenfalls durch die FIDO Alliance zertifiziert worden sein. Dabei ist die biometrische Zertifizierung [20] unabhängig von den anderen Zertifizierungen und umfasst nur die biometrische Komponente. Erst wenn diese Voraussetzungen für den Authentifikator gegeben sind, darf das angestrebte Sicherheitslevel des Authentifikators geprüft werden. In den Zertifizierungsprozess [6] sind neben dem Hersteller und der FIDO Alliance auch von der FIDO Alliance akkreditierte Sicherheitslabore involviert. Diese Labore unterstützen bei der Zertifizierung der Hard- und Software des Tokens für Level 2 und höher, um festzustellen, ob alle Sicherheitsanforderungen für das angestrebte Zertifizierungslevel erfüllt worden sind. Die Zertifizierung kann dabei sowohl von Mitgliedern der FIDO Alliance als auch von externen Herstellern für ein Produkt beantragt werden.

Abbildung 8: Zertifizierungsprozess für Authentifikatoren nach [6].

Abbildung 8 stellt die einzelnen Schritte des Zertifizierungsprozesses, wie in [6] beschrieben, dar. In der Vorbereitungsphase erfolgt die Implementierung des Tokens unter Einhaltung der Sicherheitsanforderungen des gewünschten Zertifizierungslevels. Soll das Token für Level 2 oder höher zertifiziert werden, so muss der Hersteller zudem eines der FIDO-akkreditierten Sicherheitslabore zur Prüfung auswählen. Ist die Implementierung abgeschlossen, erfolgt die funktionale Zertifizierung [19] des Tokens. Diese umfasst einen Selbsttest des Herstellers gegen ein von der FIDO Alliance bereitgestelltes Testtool sowie die Überprüfung der Interoperabilität mit den Systemen anderer Hersteller und Referenzsystemen. Wurden diese Tests bestanden, erhält das Token die funktionale Zertifizierung. Daraufhin darf der Hersteller des Tokens die Bewerbung für eine Zertifizierung des Tokens auf ein bestimmtes Sicherheitslevel einreichen. Wird

diese angenommen, beginnt die Sicherheitsbewertung durch die FIDO Alliance und ggf. durch das vom Hersteller ausgewählte Sicherheitslabor. Wird die Sicherheitsbewertung bestanden, kann die Zertifizierung des Authentifikators erfolgen. Werden später Änderungen an der zertifizierten Implementierung vorgenommen, kann zudem die erneute Durchführung einzelner Schritte notwendig sein.

Nachfolgend werden die einzelnen Zertifizierungen kurz vorgestellt.

5.1.1 Funktionale Zertifizierung

Die funktionale Zertifizierung [19] besteht aus zwei Teilen und kann für Server, Clienten und Authentifikatoren durchgeführt werden. Dabei stellt die funktionale Zertifizierung die Voraussetzung für die weitere Zertifizierung eines Authentifikators dar. Um die Zertifizierung zu erhalten, müssen ein vom Hersteller ausgeführter Selbsttest sowie ein mit anderen Herstellern organisierter Interoperabilitätstest stattfinden.

Zur Durchführung des Selbsttests, wird ein Testtool durch die FIDO Alliance bereitgestellt. Dieses Tool kann durch die Hersteller dazu genutzt werden, die Konformität der Implementierung zum Standard nachzuweisen. Das Tool protokolliert dabei die Testdurchläufe und erstellt ein Protokoll, welches die Testergebnisse nachweist. Wurden alle Tests bestanden, ist der Selbsttest erfolgreich. Anschließend darf die Implementierung des Tokens bis zum Interoperabilitätstest nicht mehr geändert werden. Neben dem Testtool führt die FIDO Alliance eine Liste von Referenzimplementierungen anderer Organisationen und Hersteller, welche zum zusätzlichen Testen der eigenen Implementierung genutzt werden kann.

Der Interoperabilitätstest muss innerhalb von 90 Tagen nach dem Datum des erfolgreich ausgeführten Selbsttests stattfinden. Dazu müssen sich die Hersteller bei von der FIDO Alliance organisierten Testing-Events registrieren. Ziel dieser Treffen ist es, die zu zertifizierenden Implementierungen der Teilnehmer untereinander auf Interoperabilität zu überprüfen. Dabei wird jede Implementierung eines Herstellers gegen jede der anderen angemeldeten Implementierungen auf Kompatibilität geprüft. Jeder dieser Tests muss durch eine Implementierung bestanden werden, um die Zertifizierung zu erhalten. Zu diesem Zweck dürfen die Teilnehmer während des Events wieder Änderungen an den Implementierungen vornehmen. Jedoch müssen in diesem Fall alle bereits abgeschlossenen Tests erneut durchgeführt werden. Zur Durchführung des Kompatibilitätstests werden im Falle von FIDO2 und U2F mindestens zwei Server und zwei Authentifikatoren benötigt. Soll die Zertifizierung eines UAF-Produktes vorgenommen werden, müssen zusätzlich zwei Clienten vorhanden sein. Stehen nicht genug Implementierungen zur Verfügung, können durch die FIDO Alliance weitere Referenzsysteme zum Test vorgegeben werden. Unter bestimmten Umständen kann die funktionale Zertifizierung zudem vertraulich, das heißt unter Ausschluss anderer Hersteller, durchgeführt werden. Gründe sind zum Beispiel geheime Implementierungen oder die Vermeidung von Betriebsspionage. Um eine vertrauliche Zertifizierung durchführen zu dürfen, müssen die entsprechenden Gründe der FIDO Alliance vorgelegt werden, welche abschließend über die vertrauliche Zertifizierung entscheidet.

Existieren zu wenig Referenzimplementierungen für den Interoperabilitätstest, kann keine funktionale Zertifizierung erfolgen. Dies kann beispielsweise bei dem Release eines neuen Standards vorkommen. In diesem Fall können die Hersteller, bei erfolgreichem Selbsttest, die Erlaubnis erhalten, das Produkt als „First Implementer“ zu bewerben. Damit ist garantiert, dass die minimalen Anforderungen an das Produkt erfüllt worden sind. Sobald ausreichend Implementierungen existieren, wird der „First Implementer“ aufgehoben und die funktionale Zertifizierung muss durchgeführt werden. In diesem Fall kann der Interoperabilitätstest On-Demand ausgeführt werden. Das heißt, der Hersteller kann seine Implementierung für Tester zugänglich machen. Sobald genug Referenzimplementierungen vorhanden sind, wird der Test anschließend durch die Tester durchgeführt und das Ergebnis an den Hersteller übermittelt.

Im Falle eines FIDO2-Tokens umfassen die Testfälle die Registrierung, die Authentifizierung und den Umgang mit (un-)bekannten Erweiterungen des FIDO2-Protokolls. Sofern für die Implementierung vorhanden, werden auch die Funktionalität des *clientPin*, die Speicherung von Resident Keys, die Erzeugung von Server Keys oder die Unterstützung mehrerer Accounts bei dem gleichen Dienst geprüft.

5.1.2 Biometrische Zertifizierung

Damit biometrische Komponenten in Authentifikatoren des Levels 3 oder 3+ verwendet werden können, müssen diese nach den Maßstäben der FIDO Alliance zertifiziert worden sein. Dazu müssen die zur biometrischen Zertifizierung geltenden Richtlinien [20] der FIDO Alliance erfüllt werden. Da diese Zertifizierung losgelöst von den anderen beiden Zertifizierungen ist, sind keine Voraussetzungen für die Durchführung gegeben. Die Tests der biometrischen Komponenten werden dabei durch ein von der FIDO Alliance akkreditiertes Sicherheitslabor ausgeführt. Zur Testdurchführung muss der Hersteller die vollständig funktionsfähige biometrische Komponente, die dazugehörigen Metadaten, wie z. B. die selbst ermittelte False Accept Rate (FAR), eigene Testergebnisse sowie die notwendige Dokumentation bereitstellen. Zudem muss der Testrahmen festgelegt werden. Auch die Bereitstellung von biometrischen Daten kann durch den Hersteller erfolgen. Anschließend kann das Labor den Test durchführen.

Sind für die Einbettung der Komponente im Authentifikator Änderungen zwingend notwendig, können diese durch den Hersteller vor Testdurchführung in einem „Allowed Integration Document“ festgehalten werden. Das Dokument wird anschließend zur Zertifizierung eingereicht und durch das Labor bewertet. Nur die in dem Dokument aufgeführten Änderungen dürfen anschließend an der zertifizierten Komponente durchgeführt werden. Änderungen, die nicht in dem Dokument festgehalten wurden, müssen erst durch eine Delta- oder Neuzertifizierung genehmigt werden.

Der Test umfasst dabei u.a. die Ermittlung der False Reject Rate (FRR) und der False Accept Rate (FAR) sowie die allgemeine Anwendbarkeit und Fehleranfälligkeit bei verschiedenen Benutzern und verschiedene Angriffe auf den jeweiligen Sensortyp [67]. Dabei sollte die FAR unter $1 : 10000$ liegen sowie die FRR unter $3 : 100$ bei einer oberen Schranke des Konfidenzintervalls von 80%.

5.1.3 Authentifikatorzertifizierung

Die Richtlinien zur Zertifizierung eines Authentifikators werden in [6] beschrieben. Voraussetzung zur Durchführung der Zertifizierung eines Authentifikators ist eine erfolgreich durchgeführte funktionale Zertifizierung. Verwendet der Authentifikator biometrische Komponenten und soll für Level 3 oder Level 3+ eingestuft werden, müssen diese zudem zuvor nach den Maßstäben der biometrischen Zertifizierung untersucht und erfolgreich zertifiziert worden sein. Für die Einstufung nach Level 1 oder 2 wird die biometrische Zertifizierung des Sensors nicht vorausgesetzt [20]. Erst wenn alle Voraussetzungen erfüllt sind, darf der Authentifikator zur Zertifizierung auf ein bestimmtes Sicherheitslevel angemeldet werden [6]. Wurde die Anmeldung akzeptiert, wird die Sicherheitsbetrachtung durchgeführt. Dazu müssen der Authentifikator sowie eine ausführliche Beschreibung dessen vorliegen. Die Überprüfung erfolgt für Level 1 und 1+ ausschließlich durch die FIDO Alliance. Ab Level 2 erfolgt die Analyse durch ein vom Hersteller gewähltes und FIDO-akkreditiertes Sicherheitslabor. Die Ergebnisse werden anschließend an die FIDO Alliance übermittelt, welche über die Zertifizierung abschließend entscheidet. Die Zertifizierung kann, wie bei der funktionalen Überprüfung, ebenfalls vertraulich durchgeführt werden. In diesem Fall darf der Hersteller nicht mit einer eventuell erreichten Zertifizierung werben, bis die Vertraulichkeit aufgehoben wurde.

Werden nach erfolgter Zertifizierung Änderungen am Authentifikator vorgenommen, muss eine Deltazertifizierung durchgeführt werden [6]. Dazu muss der Hersteller den Umfang der Änderungen evaluieren und der FIDO Alliance mitteilen. Anschließend werden die von den Änderungen betroffenen Anforderungen durch die FIDO Alliance oder das Sicherheitslabor erneut überprüft, um die Zertifizierung zu bestätigen. Dabei gibt es mehrere Gründe, bei denen eine Deltazertifizierung notwendig wird. Diese sind nicht nur mit Änderungen am Produkt verbunden, sondern beispielsweise auch mit einer neuen Version des Standards. Wird eine neue Version des Standards veröffentlicht, muss erst eine Deltazertifizierung zu der neuen Version erfolgen, um die Zertifizierung behalten zu dürfen. Auch eine vom Hersteller beantragte Erhöhung oder Senkung des Sicherheitslevels oder das Finden und Schließen von Sicherheitslücken kann zu einer Deltazertifizierung führen.

Wie in Abbildung 9 dargestellt, kann die Zertifizierung eines Tokens fünf verschiedene Zustände annehmen [6]. Neben dem Status „Aktiv“, das heißt im Zertifizierungsprozess befindlich, und dem Status „Zertifiziert“ , sind diese „Vertraulich“, „Veraltet“ und „Zurückgenommen“. Dabei kann der Zustand „Vertraulich“ nur dann erreicht werden, wenn eine vertrauliche Zertifizierung des Tokens durch die FIDO Alliance genehmigt wurde. Dieser Zustand wird solange gehalten, bis der Hersteller die Vertraulichkeit der Zertifizierung aufhebt. Ansonsten befindet sich die Zertifizierung während der Sicherheitsbewertung im Zustand „Aktiv“. Der Status „Veraltet“ wird immer dann vergeben, wenn eine neue Version des Standards, für den ein Token zertifiziert wurde, veröffentlicht wird oder nachträglich eine Verletzung der Sicherheitsrichtlinien des Standards durch das Token festgestellt wurde. Um eine neue Zertifizierung zu erhalten, können jedoch die notwendigen Änderungen am Token vorgenommen und anschließend

die Deltazertifizierung durchlaufen werden. Wird der Zustand „Veraltet“ jedoch über 180 Tage lang gehalten, weil der Hersteller nicht reagiert, wird das Zertifikat durch die FIDO Alliance zurückgenommen. Die Möglichkeit einer Deltazertifizierung wird ab diesem Zeitpunkt ausgeschlossen und das Produkt verbleibt in einem unzertifizierten Zustand. Gleiches passiert, wenn eine Verletzung der mit der FIDO Alliance geschlossenen Verträge vorliegt oder gefundene Schwachstellen und Sicherheitslücken nicht beseitigt werden. Zudem können unangekündigte und erneut geprüfte Änderungen der Implementierung eines zertifizierten Produktes zu einer Rücknahme des Zertifikats führen.

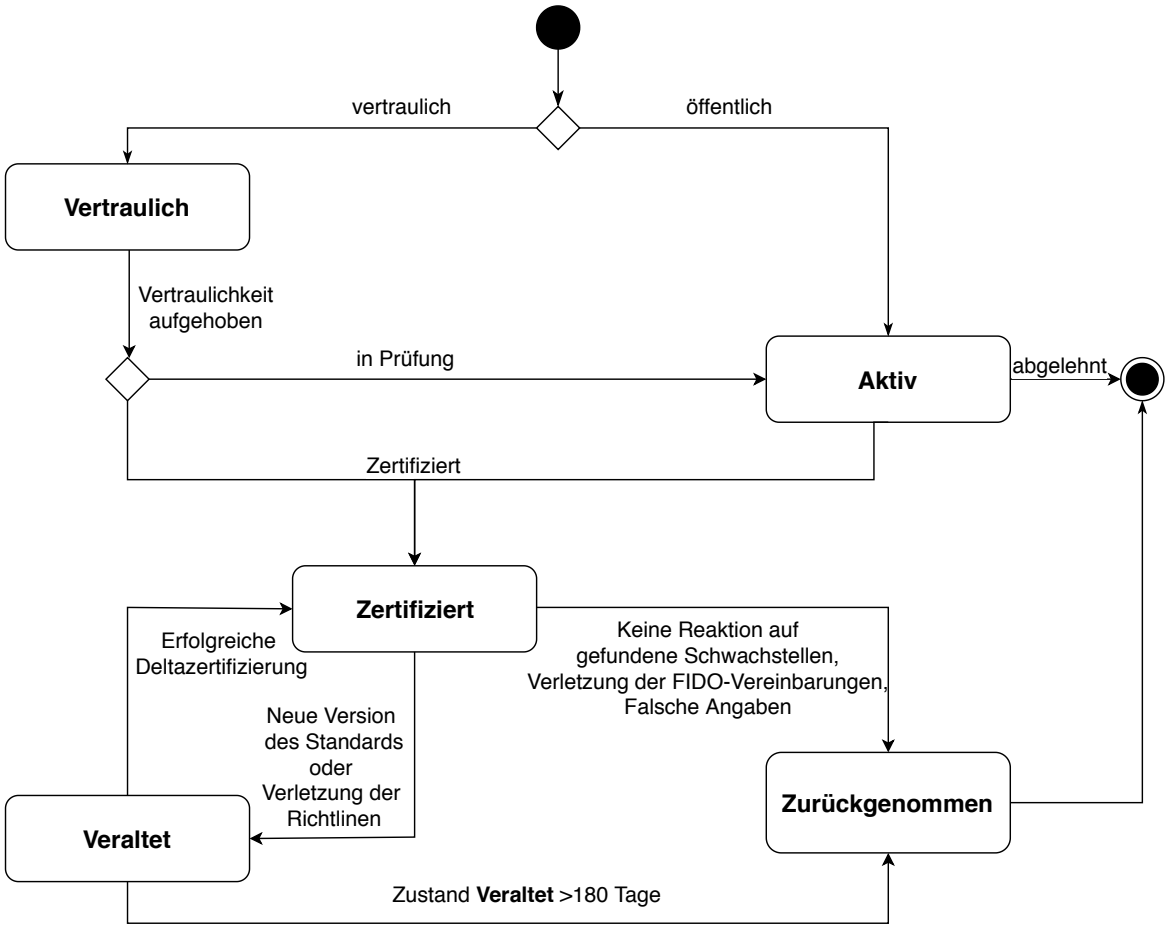

Abbildung 9: Eine Übersicht der Zustände nach [6], welche die Zertifizierung eines Produktes erreichen kann.

Die Authentifikatorzertifizierung [6] umfasst dabei die Überprüfung der in den Sicherheitsrichtlinien [68] für das angestrebte Level definierten Sicherheitsanforderungen sowie eine Analyse der eingereichten Metadaten nach [69], der verwendeten Kryptographie nach [70] und der Ausführungsumgebung nach [71] sowie Tests zur Feststellung der Angreifbarkeit des Tokens.

5.2 Zertifizierungslevel

Nachdem im vorherigen Abschnitt 5.1 der Zertifizierungsprozess sowie die einzelnen Zertifizierungen betrachtet wurden, werden in diesem Abschnitt die Zertifizierungslevel für Authentifikatoren vorgestellt. Diese werden in den Richtlinien zur Authentifikatorzertifizierung [6] sowie durch die Sicherheitsanforderungen [72] beschrieben. Die jeweiligen Sicherheitslevel verhalten sich additiv, was bedeutet, dass höhere Level die Sicherheitsanforderungen der niedrigeren Level ebenfalls erfüllen. Somit kann eine Implementierung auch mehrere FIDO-Zertifizierungen erhalten, sofern diese beantragt wurden. Zudem geht daraus hervor, dass die Sicherheit eines Tokens anhand des Levels eingestuft werden kann. Je höher das Level, desto besser ist das jeweilige Token gegen verschiedene Angriffe und Gefahren geschützt. Dies spiegelt sich auch in den Anforderungen wieder. Um Level 2+ und höher erreichen zu können, werden beispielsweise weitere Zertifizierungen der zugrundeliegenden Komponenten, wie der Ausführungsumgebung der Implementierung, nach Common Criteria (CC) EAL 4+ benötigt. Eine Übersicht über die Schutzmaßnahmen und beispielhafte Authentifikatoren wird unter [73] gegeben. Im Folgenden wird ein kurzer Überblick über die einzelnen Sicherheitslevel und deren Schutzziele basierend auf den Informationen aus [6, 72, 73] gegeben.

Level 1 Bei der Zertifizierung auf Level 1 wird die Implementierung der Sicherheitsmaßnahmen des Tokens bewertet. Tokens dieses Levels schützen gegen Phishing, Man-in-the-Middle-Angriffe und den Verlust von Anmeldeinformationen. Dabei spielt die Soft- und Hardware, auf der das Token läuft, keine Rolle. Ein beispielhaftes Token wäre eine App, die auf Smartphone-eigene, nicht zertifizierte Soft- oder Hardware zugreift.

Level 1+ Wird ein Token auf Level 1+ zertifiziert, bietet es neben den für Level 1 geltenden Schutzmaßnahmen auch einen Schutz gegen die Kompromittierung durch das Betriebssystem, auf dem das Token läuft. Dies könnte z.B. durch die Anwendung von White Box Kryptographie erfolgen.

Level 2 Zur Zertifizierung auf Level 2 wird das Token auf seinen Schutz gegen groß angelegte Softwareangriffe untersucht. Ab dieser Stufe muss für die Untersuchung des Tokens ein FIDO-akkreditiertes Sicherheitslabor beauftragt werden. Level 2 zertifizierte Tokens bieten einen Schutz gegen die Kompromittierung des Betriebssystems, auf dem die Implementierung läuft. Dazu muss die Verwendung einer der in [71] genannten restriktiven Ausführungsumgebungen (ROE) unterstützt werden. Zudem darf die Implementierung nur die in [70] genannten kryptographischen Algorithmen verwenden. Ein Beispiel für ein solches Token wäre Implementierung einer vertrauenswürdigen Applikation innerhalb einer unzertifizierten TEE.

Level 2+ Level 2+ erweitert die Anforderungen dahingehend, dass ein größerer Schutz gegen großangelegte Softwareangriffe geboten werden muss als bei Level 2. Dies wird dadurch erreicht, dass das TEE, welches verwendet wird, nun nach Common Criteria (CC) o.ä. zertifiziert worden sein muss.

Level 3 Für das Level 3 muss nachgewiesen werden, dass das Token gegen grundlegende Soft- und Hardwareangriffe mit erhöhtem Aufwand geschützt ist. Als Orientierung gelten die in AVA_VAN.3 durch die CC festgehaltenen oder äquivalente Angriffe. Ein Token, dass Level 3 zertifiziert wurde, schützt dabei gegen Angriffe auf die Platine durch z. B. das Abhören der Leiterbahnen. Dafür wird u.a. die Verwendung von verschlüsseltem RAM oder das Eingießen der Platine in eine solide Komponente, wie beispielsweise Epoidyharz, empfohlen. Ein Beispiel für ein Level 3 zertifiziertes Token wäre ein gegen Manipulationen geschütztes Token, welches auf einer simplen CPU mit CC-zertifiziertem Betriebssystem läuft. Das Token könnte beispielsweise als USB-Token implementiert werden.

Level 3+ Level 3+ stellt das höchste Level der Zertifizierung dar. Tokens, die auf dieses Level geprüft werden, müssen sowohl einen Schutz gegen fortgeschrittene als auch einen Schutz gegen aufwändige Soft- und Hardwareangriffe bieten. Beispiele für solche Angriffe werden in der AVA_VAN.4 genannt. Tokens, die Level 3+ erreichen, schützen dabei gegen Angriffe auf die verbauten Microchips. Dazu muss ein Schutz gegen die Injektion von Fehlern oder gegen invasive Attacken auf die Chips gewährleistet werden. Ein Beispiel wären Tokens in USB-Form, die auf einem CC-zertifizierten sicheren Element implementiert sind.

6 Umsetzung

Ziel der Arbeit war es, ein funktionsfähiges, biometrisches FIDO2-Token zu entwickeln, welches auf einer Smartcard mit Fingerabdrucksensor (im Folgenden auch: Fingerabdruckkarte) verwendet werden kann. Die FIDO2-Funktionalität wurde dabei in Form eines Java Card Applets implementiert, wobei die Java Card Version 3.0.2 verwendet wurde. Zudem dient als Ausführungsumgebung eine von der Bundesdruckerei GmbH entwickelte Fingerabdruckkarte, welche für die Anfertigung der Arbeit zur Verfügung gestellt wurde.

Im folgenden Kapitel wird auf die Grundlagen zu Smartcards sowie Biometrie eingegangen. Daraufhin wird die Architektur und Ausführungsumgebung der Fingerabdruckkarte vorgestellt. Abschließend werden Designentscheidungen der Implementierung dargelegt.

6.1 Smartcards

Die Grundsteine zur Entwicklung der Smartcards wurden bereits 1968 durch Jürgen Dethloff und Helmut Grötrupp gelegt. Diese entwickelten die Idee der Chipkarte und ließen sich diese anschließend patentieren. Mit der Zeit entwickelten weitere Forscher neue Ideen, woraus die heutigen Smartcards hervorgingen. Smartcards sind Plastikkarten, welche in vielen verschiedenen Größen und mit unterschiedlichen Eigenschaften ausgestattet sind. Sie finden beispielsweise Anwendung als SIM-Karten für Handys, Krankenversicherungskarten, Kreditkarten und Girocards, aber auch als Personal- und

Studierendenausweise sowie andere Authentifikationsmittel. Die Anwendung kann dabei kontaktbehaftet oder kontaktlos erfolgen. Dabei werden in der ISO 7816 verschiedene Eigenschaften, wie z.B. das Design und die Kommunikationsprotokolle von Smartcards standardisiert. [7, Kapitel 1] [27, Kapitel 11]

Chipkarten werden, je nach Eigenschaften, in verschiedene Kategorien eingeteilt. Ein typisches Kriterium ist die Kommunikationsmethode mit dem Lesegerät. Kontaktbehaftete Karten können über auf der Außenseite angebrachte Kontakte mit dem Lesegerät kommunizieren. Kontaktlose Karten kommunizieren via NFC mit dem Lesegerät. Dabei wird der von der Karte benötigte Strom für den Betrieb über ein Magnetfeld des Lesegeräts induziert. Hierbei schließen sich die beiden Kommunikationsmethoden nicht gegenseitig aus. Heutige Girokarten bieten z.B beide Kommunikationsmethoden an, welche je nach Bedarf, Vorlieben und Möglichkeiten durch den Benutzer angewendet werden können. [7, Kapitel 2]

Neben der Unterscheidung in kontaktbehaftete und kontaktlose Chipkarten wird auch die Unterscheidung in Speicherkarten und Mikroprozessorkarten vorgenommen. Speicherkarten dienen dabei lediglich zur Speicherung von vorher festgelegten Daten über den Benutzer und können in der Anwendung nur vorher festgelegte Instruktionen auf den Daten ausführen. Die Daten werden während der Herstellung in einen ROM geschrieben, sodass diese nicht verändert werden können. Eine Wiederverwendung bei Änderung der Daten ist somit nicht möglich. [7, Kapitel 2] [27, Kapitel 11]

Mikroprozessorkarten, im Folgenden Smartcards genannt, sind hingegen Karten, die über eine eigene Recheneinheit verfügen. Dabei sind Prozessoren mit bis zu 32-Bit möglich. Neben dem Prozessor sind auch ein ROM mit einer Größe von bis zu 320KB für kartenspezifische Daten sowie Applikationen und das Betriebssystem, ein RAM mit bis zu 16 KB Speicher als Arbeitsspeicher und ein EEPROM zum Speichern von weiteren Programmen und Daten verbaut. Der EEPROM kann bis zu 400KB Daten halten und mindestens 100 000 mal beschrieben werden. Neben den Standardkomponenten können auch weitere Co-Prozessoren und Sensoren in den Schaltkreis einer Karte integriert werden. Smartcards können aufgrund ihrer Struktur eine oder mehrere Anwendungen beinhalten, welche Daten generieren, speichern und verarbeiten können. Wird ein frei programmierbarer Mikroprozessor verwendet, können die Programme zudem während des Betriebs durch autorisierte Kartenlese- und Schreibgeräte neu programmiert oder Updates der darauf befindlichen Programme durchgeführt werden. [7, Kapitel 2] [27, Kapitel 11]

Im Folgenden wird auf die Betriebssysteme von Smartcards, insbesondere auf Java Card basierende, eingegangen. Daraufhin wird die Kommunikation einer Smartcard betrachtet. Abschließend wird die Sicherheit von Smartcards diskutiert.

6.1.1 Betriebssysteme

Das Betriebssystem einer Smartcard wird im ROM-Bereich der Karte eingebrannt. Die Aufgabe des Betriebssystems ist es, die grundlegende Funktionalität für Anwendungen bereitzustellen und die Anwendungen selbst auszuführen. Zu den grundlegenden Funktionalitäten eines Betriebssystems gehören unter anderem die Abwicklung der

Kommunikation, die Durchführung der Speicherzugriffe, die Bereitstellung von Sicherheitsmechanismen zum Zugriffsschutz, wie beispielsweise die PIN-Eingabe einer Karte und deren Verwaltung, sowie der Schutz sensibler Daten, z.B. durch die Isolierung der auf der Karte ausgeführten Anwendungen. Dazu wird z.B. jeder Speicherzugriff auf den EEPROM durch das Betriebssystem auf Legitimität und Autorisierung überprüft. Weiterhin bieten viele Smartcard-Betriebssysteme verschiedene native Bibliotheken an, welche unter anderem effiziente Implementierungen bekannter kryptographischer Algorithmen für das entsprechende System bereitstellen. Die Anzahl und Art der bereitgestellten nativen Bibliotheken und der darin enthaltenen Algorithmen und Abstraktionen ist dabei je nach Betriebssystem unterschiedlich. [7] [27, Kapitel 11]

Betriebssysteme können auf verschiedenen Programmiersprachen, wie z.B. Java basieren. Die für Smartcards entwickelte Programmiersprache Java Card stellt dabei eine reduzierte Version der Java VM dar. Dies ist auf die geringen Speicherkapazitäten einer Smartcard zurückzuführen. Die Smartcard unterstützt nur die notwendigsten Java-Features, wie beispielsweise die kleinen primitiven Datentypen *short*, *boolean* und *byte* und eindimensionale Arrays. Einige Implementierungen bieten auch ein 32-bit Integer an. Weiterhin sind Strukturen wie Vererbung, Interfaces und Packages im Umfang enthalten und auch die Fehlerbehandlung wurde übernommen. Features wie Nebenläufigkeit, Garbage Collection, große primitive Datentypen wie *long*, *double* und *float* sowie komplexere Typen wie *strings* werden nicht unterstützt. [7, Kapitel 3]

Wie Chen [7, Kapitel 3] beschreibt, bietet die Java Card Runtime Environment (JCRE) der Smartcard eigene Features an. So werden alle auf der Smartcard erzeugten Objekte standardmäßig im persistenten Speicher, d. h. im EEPROM, angelegt. Da die Zugriffe jedoch zeitaufwändig sind, besteht die Möglichkeit, direkt auf den RAM zuzugreifen und Objekte dort ablegen und verwalten zu können. Die dort hinterlegten Objekte werden jedoch nach dem Stromverlust der Karte wieder gelöscht. Weiterhin wird die Atomarität jeder Schreibinstruktion garantiert. Zudem wird das Konzept der Transaktionen für mehrere Schreibzugriffe umgesetzt. Sollen mehrere Schreibzugriffe erfolgen, von denen nur alle oder gar keine persistent gespeichert werden sollen, kann der Entwickler eine entsprechende Transaktion verwenden. Abschließend werden durch die JCRE die einzelnen Applets auf einer Smartcard von den anderen Applets isoliert. Jede Karte darf nur auf ihre eigenen Daten zugreifen und nicht aus den Bereichen der anderen Applets lesen und schreiben. Da dies in manchen Fällen dennoch notwendig ist, können von Java Card bereitgestellte Schnittstellen verwendet werden. Beispielhaft sei die Situation erwähnt, dass ein Applet der Smartcard für die Überprüfung des Fingerabdrucks zuständig ist. Andere Applets implementieren verschiedene Protokolle, die nur dann ausgeführt werden sollen, wenn der entsprechende Fingerabdruck vorhanden ist. In diesem Fall soll nur das Fingerabdruckapplet über die Samples des Fingerabdrucks verfügen, die anderen Anwendungen müssen jedoch dem Applet mitteilen können, dass der Fingerabdruck aufgenommen und entsprechend verifiziert werden muss. Abschließend benötigen die Anwendungen das Ergebnis der Überprüfung.

Zur Entwicklung von Applets wird eine Java Card API angeboten, welche die zuvor beschriebenen Features wie Transaktionen, Speicherzugriffe, Schnittstellen aber auch kryptographische Verfahren nativ bereitstellt. Damit eine Java Card Applikation auf

der Smartcard installiert werden kann, muss diese zunächst für die auf der Smartcard ausgeführte JCRE kompiliert werden. Anschließend kann die kompilierte Applikation auf der Smartcard installiert werden. Dabei erfolgt die Installation transaktional, tritt also ein Fehler auf, werden alle bereits auf der Karte hinterlegten Daten wieder gelöscht. Für die Installation können dem Applet Argumente mitgegeben werden, welche in der Installationsroutine verwendet werden. Dies können z.B. Zertifikate, aber auch andere Daten sein. Während der Installation werden zudem alle zur Laufzeit benötigten Objekte instanziiert. [7, Kapitel 3]

6.1.2 Kommunikation

Die Kommunikation zwischen Kartenleser und Smartcard erfolgt sowohl bei kontaktbehafteten als auch bei kontaktlosen Verfahren über Application Protocol Data Units (APDUs). Hierbei handelt es sich um Pakete, die durch die Smartcard empfangen und verschickt werden können. Durch die Smartcard empfangene APDUs werden auch als Befehls-APDUs bezeichnet, von der Smartcard versendete APDUs als Antwort-APDUs. Wie in Abbildung 10 ersichtlich, unterscheiden sich Befehle und Antworten in ihrer Struktur. Während Befehle festlegen müssen, welche Operation durch die Smartcard ausgeführt werden soll und gegebenenfalls entsprechende Parameter übermitteln, reicht für die Antwort die Übermittlung des Ergebnisses der Operation aus. Beide Befehle haben gemein, dass nur der *Body* optional ist. Dieser muss somit keine Daten enthalten. [7, Kapitel 2]

Abbildung 10 (a) zeigt dabei die Struktur einer Befehls-APDU. Diese enthält das Byte *CLA* zur Festlegung der Operationsklasse, das Byte *INS* zur Festlegung der auszuführenden Instruktion der Klasse sowie zwei Parameterbytes *P1* und *P2* um weitere Angaben für die auszuführenden Operationen bereitzustellen. Müssen weitere Daten zur Verwendung durch die Instruktion ausgetauscht werden, kann dies über den *Body* erfolgen. Dazu beschreibt das L_c-Byte die Länge der darauffolgenden Daten und begrenzt die maximale Anzahl an Datenbytes auf 256, da der Wert 0 für L_c als 256 gewertet wird. Überschreitet die zu übermittelnde Nachricht die maximale Länge von 256 Bytes, muss diese in mehrere APDUs aufgeteilt werden. Wird zudem eine Antwort bekannter Länge erwartet, kann die Länge der Antwort über das L_e-Byte spezifiziert werden. [2] [7, Kapitel 2, 8]

Da die Smartcard nur die Daten der ausgeführten Berechnung bzw. den Status der Berechnung an den Kartenleser zurückgeben muss, ist die Struktur einer Antwort simpler gestaltet, wie in Abbildung 10 (b) erkenntlich. Dabei werden, sofern vorhanden, die berechneten Daten in den *Body* der Antwort geschrieben. Die Länge der Daten ist, wie bei der Befehls-APDU, auf 256 Bytes beschränkt. Auch in diesem Fall müssen längere Nachrichten in mehrere APDUs aufgeteilt werden. Abschließend werden zwei Status-Bytes SW_1 und SW_2 angehängt. Diese geben den Status der Operation wieder – 0x9000 steht dabei für den Erfolg einer Instruktion, ein anderer Wert gibt einen Fehler oder anderen Zustand an. [7, Kapitel 2, 8]

Neben den in Abbildung 10 dargestellten APDUs, auch als Short Length APDUs bezeichnet, werden durch die ISO 7816 auch Extended Length APDUs definiert. Diese

Head				Body		
CLA	INS	P_1	P_2	L_c	Daten	L_e

(a) Befehls-APDU

Body	Tail	
Daten	SW_1	SW_2

(b) Antwort-APDU

Abbildung 10: Struktur einer Befehls- und einer Antwort-APDU nach [7, Kapitel 2].

kodieren die Länge des Datenteils und die Länge der erwarteten Antworten mit jeweils 2 Bytes. Somit ergibt sich eine maximale Länge von 65.536 Bytes für die jeweiligen Datenteile von Befehl- und Antwort-APDU, da auch hier die Länge 0 als 65 536 interpretiert wird [2].

6.1.3 Sicherheitsbetrachtung

Durch Eckert [27, Kapitel 11] werden verschiedene Sicherheitsmaßnahmen des Designs von Smartcards vorgestellt. Trotz der Sicherheitsmaßnahmen existieren verschiedene Angriffe auf Smartcards, welche ebenfalls durch Eckert beschrieben werden. Diese basieren meist auf einer physikalischen Einwirkung auf die Smartcard, sodass Fehler im Betrieb der Hard- oder Software zu Veränderungen und Fehlern führen, wodurch geheime Schlüssel vorhersagbar werden. Dies kann z.B. in Form der Veränderung der Umgebungstemperatur der Smartcard oder durch die kurzzeitige Erhöhung der Spannung erfolgen. Um sich dagegen zu schützen, werden verschiedene Sensoren in den Mikrocontroller integriert, welche eine Veränderung einiger physikalischer Eigenschaften, wie Temperatur und Spannung, erkennen. Werden Änderungen erkannt, kann der Mikrocontroller entsprechend reagieren.

Neben physikalischen Angriffen auf die Smartcard sind auch physische Angriffe unter Verwendung verschiedener Techniken und Geräte möglich. Hierbei wird versucht, die interne Kommunikation oder Speicherbereiche auszulesen oder zu manipulieren. Die dagegen verwendeten Schutzmaßnahmen sind sehr vielfältig. Zum Beispiel werden interne Busse des Prozessors nicht nach außen geleitet und deren Anordnung zufällig und individuell durchgeführt, um die Kontaktierbarkeit der Busse zu unterbinden bzw. bei Erfolg das Auslesen der Funktion des Busses zu erschweren. Einige Chips verschlüsseln zusätzlich die Kommunikation zwischen den einzelnen On-Chip-Komponenten, um die Daten ebenfalls gegen das Auslesen zu schützen. Weiterhin werden die ROM-Bereiche in tieferen Siliziumschichten eines Chips angesiedelt, um die Modifikation sowie das Reverse Engineering des Betriebssystems zu erschweren. Zudem wird der ROM-Code ionenimplantiert, wodurch das Auslesen, z.B. durch Mikroskope oder mittels Infrarot, unterbunden wird. Damit die elektromagnetische Strahlung eines Chips nicht aufgezeichnet werden kann, wird der Chip durch spezielle Schutzschichten, wie z.B. einer dünnen leitenden Metallschicht und einer UV-Strahlungsabweisenden Schicht, isoliert. Wird die leitende Schicht beschädigt, führt dies zur kompletten Zerstörung des Chips

aufgrund der unterbrochenen Stromversorgung. Die UV-Strahlungsabweisende Schicht soll die Löschung des EEPROMs verhindern. Um eine geringfügige Beschädigung der Schichten zu erkennen, können ebenfalls Sensoren verwendet werden. [27, Kapitel 11]

Weiterhin stellt der bereits in Abschnitt 2.2.3 erwähnte Nebenkanalangriff über die Analyse der Stromaufnahme der Hardware, während der Berechnung eines Algorithmus, eine Gefahr für Smartcards dar. Um diesen Nebenkanalangriff zu verhindern, kann beim Design des Mikrochips versucht werden, die Energieaufnahme unabhängig von den durch den Chip verarbeiteten Daten und Instruktionen zu halten. Dazu könnten z.B. ein Spannungsregler und Widerstände verbaut werden. Auch bei einer ungünstigen Implementierung von Algorithmen, wie beispielsweise einer unvorteilhaften Implementierung des PIN-Vergleichs, kann über die Energieaufnahme die Erhöhung des Fehlversuchszählers vorausgesagt werden. Bei der Implementierung sollte daher darauf geachtet werden, dass z.B. der Fehlversuchszähler inkrementiert wird, bevor der Vergleich stattfindet. Geschieht die Erhöhung erst nach dem Abgleich, könnte dies über die Stromaufnahme erkannt und der Vorgang noch vor der Erhöhung des Zählers abgebrochen werden. [27, Kapitel 11]

Doch nicht nur die interne Kommunikation kann für einen Angreifer von Interesse sein. Auch die nach außen übermittelten Daten stellen ein potentielles Angriffsziel dar. Hier sollten Techniken wie Verschlüsselungen, Signaturen und MAC-Berechnungen zum Einsatz kommen, um die Übertragung zwischen Lesegerät und Karte abzusichern und die Integrität der Daten zu wahren. Abschließend seien die Maßnahmen zur Fälschungssicherheit und zum Schutz vor unautorisierter Benutzung von Chipkarten genannt, welche häufig durch das Aufdrucken personenbezogener Daten wie Name, Vorname oder ein Foto des legitimen Nutzers ergriffen werden. [27, Kapitel 11]

6.2 Biometrie

Biometrische Eigenschaften stellen eine der in Kapitel 2 vorgestellten drei Kategorien dar, aus denen Verfahren zur Authentifizierung gewählt werden können. Dabei beschreiben biometrische Merkmale die Physiologie oder das Verhalten einer natürlichen Person und ermöglichen so die eindeutige Identifizierung des Besitzers des Merkmals. Durch physiologische Merkmale werden statische Eigenschaften des Nutzers beschrieben. Beispiele für diese Gruppe sind Fingerabdrücke, die Iris, die Retina oder das Gesicht. Dynamische Merkmale bilden das Verhalten des Nutzers ab, dazu zählen die Stimme, das Tippverhalten oder die Dynamik der handschriftlichen Unterschrift. [27, Kapitel 10]

Sowohl statische als auch biometrische Merkmale müssen dabei einige Anforderungen erfüllen, um für die Authentifizierung benutzt werden zu können. Diese umfassen zum einen die Universalität, die Eindeutigkeit sowie die Beständigkeit des Merkmals. Zudem muss das Merkmal quantitativ erfassbar und die Erfassung performant möglich sein. Auch die Akzeptanz zur Benutzung muss gegeben sein und das Merkmal sollte eine hohe Fälschungssicherheit aufweisen. Jedoch erfüllen die verschiedenen biometrischen Merkmale diese Anforderungen unterschiedlich gut. So kann z.B. die quantitative Erfassbarkeit nicht bei jedem Menschen gleich gut ausgeprägt sein, sodass das biometri-

sche Merkmal zwar existiert, aber aufgrund einer zu geringen Anzahl an Charakteristika nicht für die Authentifizierung genutzt werden kann. [27, Kapitel 10]

Damit ein Merkmal zur Authentifikation benutzt werden kann, muss dieses zuerst durch das Gerät erfasst und auf die im Merkmal enthaltenen charakteristischen Eigenschaften analysiert werden. Das Ergebnis wird als Referenz (auch: Template) auf dem System selbst hinterlegt. Möchte man sich anschließend authentifizieren, muss das Merkmal erneut durch den Sensor eingelesen werden. Danach erfolgen die Analyse nach den Eigenschaften des Merkmals und ein Abgleich mit der auf dem Gerät hinterlegten Referenz. Anders als bei Passwörtern ist hier jedoch eine vollständige Übereinstimmung der Messwerte schwer zu gewährleisten. Aufgrund verschiedener Umwelteinflüsse, Verletzungen oder Krankheiten kann die Physiologie oder das Verhalten des Benutzers eingeschränkt sein. Stattdessen wird durch das System ein Schwellwert festgelegt, ab dem genügend Übereinstimmung vorliegt, sodass die Authentifikation erfolgreich ist. Der Vergleich gegen einen definierten Referenzwert auf dem Gerät wird auch als Verifikation bezeichnet. Diesem steht die Identifikation gegenüber, bei dem der aktuell erhobene Wert gegen eine Datenbank geprüft wird, indem nach einem passenden Referenzmuster innerhalb der Datenbank gesucht wird. [27, Kapitel 10]

Nachfolgend wird der Fingerabdruck als biometrisches Merkmal genauer betrachtet, sowie eine Sicherheitsbetrachtung von biometrischen Verfahren durchgeführt.

6.2.1 Fingerabdruck

Seit den 1960er Jahren findet die Analyse und der Abgleich von Fingerabdrücken technisch statt. Damals wurde das Verfahren vom FBI zur Aufklärung und Bekämpfung von Straftaten verwendet. Dazu nutzt man die Eigenschaft, dass die meisten Personen einen Fingerabdruck besitzen und dass dieser in jedem Fall einzigartig und unveränderlich ist. Eine Veränderung des Fingerabdrucks kann dabei nur mutwillig oder durch Fremdeinwirkung erfolgen. [27, Kapitel 10]

Charakteristisch für den Fingerabdruck sind dabei die in ihm ausgeprägten Minutien. Minutien sind kleine Unterbrechungen der Papillarleisten, welche über die Fingerkuppen verlaufen. Die Minutien können in Form von Verwirbelungen, Gabelungen, Inseln oder Eckpunkten auftreten und bilden anhand ihrer Art und Ausrichtung ein einzigartiges Muster. Dieses Muster kann analysiert und für einen späteren Vergleich als Referenz hinterlegt werden. Auch wenn der Fingerabdruck selbst alle Anforderungen an ein biometrisches Merkmal erfüllt, treten hier gegebenenfalls Probleme in der quantitativen Erfassbarkeit auf. Da die Anzahl an verwertbaren Minutien von Mensch zu Mensch stark variieren kann, kann es vorkommen, dass einige Personen nicht den Schwellwert an notwendigen Charakteristika des Abdrucks überwinden können. In diesem Fall kann der Fingerabdruck aufgrund einer zu geringen Anzahl an Minutien nicht für das Verfahren verwendet werden. Weitere Charakteristika des Fingerabdrucks sind zudem der Abstand der Papillarleisten zueinander sowie die Porenstruktur. [27, Kapitel 10]

Um den Fingerabdruck aufzunehmen, können verschiedene Sensoren verwendet werden. Neben optischen Sensoren, die ausschließlich ein 2D-Bild des Fingerabdrucks aufnehmen, existieren auch kapazitive und thermische Sensoren sowie Druck- oder

Ultraschallsensoren. Je nach Sensor können dabei weitere Eigenschaften aufgenommen werden, wie beispielsweise die Temperatur des Fingers. Weiterhin ist anstelle der zweidimensionalen Erfassung des Fingerabdrucks auch die Aufnahme der dreidimensionalen Struktur des Fingerabdrucks möglich. Anhand des aufgenommen Bildes können anschließend die Minutien extrahiert und analysiert werden. Abschließend wird ein Abgleich des Musters mit der Referenz vorgenommen, indem Nachbarschaftsbeziehungen ausgewertet werden. Dazu wird die Lage der Minutien zueinander bestimmt, wobei Abstand und Winkel der Minutien von Interesse sind. Wird der Schwellwert der übereinstimmenden Minutien anhand dieser Beziehungen überschritten, wird der Fingerabdruck als gleich angesehen. [27, Kapitel 10]

6.2.2 Sicherheitsbetrachtung

Wie bei jedem Verfahren, bieten biometrische Merkmale Vor- und Nachteile in der Nutzung. So sind die enge Bindung des Merkmals an den Nutzer und die Eindeutigkeit ein klarer Vorteil für die sichere und eindeutige Identifikation einer Person. Diese Eigenschaften können jedoch in verschiedenen Szenarien zu Problemen führen. Beispielsweise kann die missbräuchliche Aufnahme und der darauffolgende Abgleich biometrischer Eigenschaften dazu führen, dass Menschen an verschiedenen Orten, die sie besuchen, identifiziert werden können. Auf diese Weise lassen sich gezielt Bewegungsprofile einzelner oder mehrerer Menschen erheben. Weiterhin können über biometrische Sensoren und Analysen neben dem eigentlichen Merkmal viele weitere Daten über eine Person erhoben und ermittelt werden. Hierzu zählen unter anderem der aktuelle Gesundheitszustand oder der Stress, den eine Person derzeit verspürt. Um diese Gefahren der missbräuchlichen Nutzung zu unterbinden, sollte darauf geachtet werden, dass erhobene Referenzwerte nicht mit der natürlichen Person in Verbindung gebracht, sondern z. B. pseudonomisiert gespeichert werden. [27, Kapitel 10]

Ein weiteres Problem der starken Bindung der biometrischen Merkmale an die natürliche Person stellt die Bindung selbst dar. Möchte ein Angreifer in sensible Bereiche eindringen, die durch Biometrie gesichert werden, benötigt er die Merkmale einer zugangsberechtigten Person. Diese kann der Angreifer durch Gewalt, Erpressung oder Entführung an sich bringen. Ansätze diese Gefahr zu mindern, stellen die Lebend-Erkennung von Sensoren oder das Hinterlegen von biometrischen Notfall-Referenzdaten dar. Zur Lebend-Erkennung kann beispielsweise die Temperatur des benötigten Merkmals, wie dem Finger, ausgewertet werden. Notfall-Referenzdaten beschreiben bei der Personalisierung hinterlegte Referenzdaten, die nur dann verwendet werden, wenn eine erpresserische oder gewaltsam erzwungene Authentifizierung der Person vorliegt. Das System kann somit den Zustand der Person erkennen und entsprechende Maßnahmen ergreifen. [27, Kapitel 10]

Auch die Unveränderlichkeit der biometrischen Merkmale kann Probleme hervorrufen. Gelingt es einem Angreifer die Referenzdaten eines Merkmals eines Benutzers zu kompromittieren, kann dies im schlimmsten Fall einen kompletten Ausschluss des Benutzers für das Authentifizierungsverfahren bedeuten. Da der Angreifer nun im Besitz der Referenzdaten ist, kann dieser bei Bedarf die Daten zur Authentifizierung

wieder einspielen. Somit kann er Zugang zu allen Bereichen, Geräten und Diensten der Person erhalten, die durch das Merkmal geschützt worden sind. Weiterhin stellt die Unveränderlichkeit auch dann ein Problem dar, wenn einem Angreifer der Austausch der hinterlegten Referenzdaten durch seine eigenen Daten gelingt. Auf diese Weise kann der Angreifer selbst Zugriff zu dem geschützten Bereich erhalten, der autorisierte Benutzer wird jedoch im schlimmsten Fall auf Dauer aus dem Bereich ausgeschlossen. Dies ist darauf zurückzuführen, dass aufgrund der falschen Referenzdaten keine erfolgreiche Authentifizierung des legitimen Benutzers mehr möglich ist. [27, Kapitel 10]

Abschließend seien hier die verwendeten Sensoren und Algorithmen zum Abgleich der biometrischen Merkmale selbst erwähnt. So ist der Schwellwert der benötigten Übereinstimmungen zwischen aktueller Aufnahme und Referenzwert mit Bedacht zu wählen. Wird dieser Wert zu niedrig festgelegt, um beispielsweise die Effizienz des Verfahrens zu erhöhen oder mehr Benutzern den Zugang zum Verfahren zu ermöglichen, kann dies zu falsch positiven Abgleichen führen, sodass unberechtigte Benutzer Zugang zum System erhalten. Wird der Wert jedoch zu hoch angesetzt, besteht das Problem, dass selbst legitime Benutzer aus dem System ausgeschlossen werden könnten, da z.B. nicht genügend Charakteristika des biometrischen Merkmals ausgeprägt sind oder der aktuelle Gesundheitszustand zu temporären Veränderungen des Merkmals führt. Das Maß zur Ermittlung der falsch positiven Abgleiche ist die False Acceptance Rate (FAR), das zur Ermittlung der fälschlich abgewiesenen Nutzer die False Rejection Rate (FRR). Hierbei möchte man beide Raten so klein wie möglich halten. Für sehr sensible Bereiche ist eine höhere FRR jedoch annehmbar. Dem im vorherigen Abschnitt 6.2.1 vorgestellten Beispiel des Fingerabdrucks folgend, spielt neben dem Verfahren auch die Qualität der Sensoren eine maßgebliche Rolle in der Sicherheit des Verfahrens. Da Fingerabdrücke öffentliche Merkmale sind, die durch den Benutzer zu verschiedenen Zeitpunkten an verschiedenen Orten verteilt werden, kann ein Angreifer diese leicht erbeuten und Nachbildungen des Fingerabdrucks anfertigen. Je nach Qualität des Sensors können Nachbildungen verschiedenen Aufwands zum Erfolg führen. So kann zum Beispiel ein Klebefilm-Abdruck des Fingerabdrucks ausreichen, um sich Zugang zum System zu verschaffen, oder aber ein Gummi-Finger benötigt werden. Diese Problematik lässt sich auch auf andere Verfahren, wie den Iris-Scan, übertragen. [27, Kapitel 10]

6.3 Fingerabdruckkarte

Die Fingerabdruckkarte der Bundesdruckerei GmbH dient als Ausführungsumgebung des im Rahmen der Arbeit implementierten FIDO2-Applets. Bei der Fingerabdruckkarte handelt es sich um eine kontaktlose Smartcard mit integriertem Fingerabdrucksensor zur Authentifizierung des Benutzers gegenüber der Karte. Dabei wird die Funktionalität der Fingerabdruckverifikation bereits durch die Karte bereitgestellt.

Abbildung 11 zeigt die Architektur der Karte. Die Karte umfasst ein sicheres Element, einen zusätzlichen Mikrocontroller, einen Fingerabdrucksensor sowie eine NFC-Antenne. Das auf der Fingerabdruckkarte verbaute SE beinhaltet dabei das Java Card-Betriebssystem JCOP3 [74], welches nach CC EAL5+ zertifiziert ist, und die auf der Karte auszuführenden sicheren Anwendungen. Als SE wird der NXP SmartMX2

P60 [75] verwendet. Dieser besitzt in der verbauten Fassung ca. 2 KByte RAM und 128 KByte EEPROM für die Speicherung von Daten und Programmen. Das SE ist dabei nach CC EAL 6+ zertifiziert.

Neben dem SE ist ein zusätzlicher Mikrocontroller (MCU) des Typs STM32L476 [76] verbaut. Dieser enthält 128 KByte RAM sowie 1 MB EEPROM. Der STM32L476 ist für die Kommunikation zwischen den einzelnen Komponenten der Karte zuständig. Zusätzlich ist ein Fingerabdrucksensor des Typs FPC1020 [77, 78] verbaut, um das Template des Fingers für Referenzdaten und den späteren Abgleich aufzunehmen. Der Sensor ist ein kapazitiver Drucksensor, welcher theoretisch ein 3D-Bild des Fingerabdrucks erstellt, das Ergebnis entspricht jedoch eher einem 2D-Bild. Das Bild wird dabei in 256 Graustufen dargestellt. Der für den Fingerabdruckvergleich verwendete Algorithmus gibt zudem eine FAR von 1 : 1000 und eine FRR von 2 : 100 an, wurde für die Fingerabdruckkarte jedoch nicht empirisch überprüft. Die NFC-Antenne dient der drahtlosen Stromversorgung der kompletten Karte sowie der Kommunikation mit dem NFC-Lesegerät.

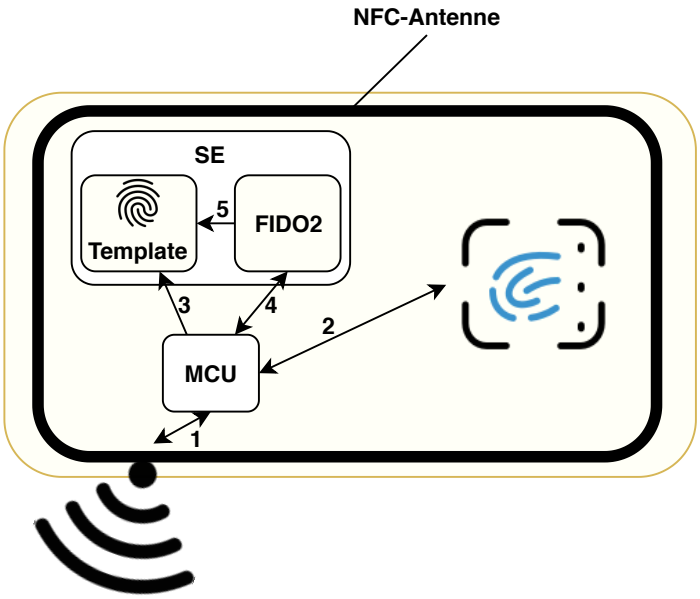

Abbildung 11: Architektur der Fingerabdruckkarte.

In Abbildung 11 wird neben den verbauten Komponenten ebenfalls der Kommunikationsablauf der Komponenten im Betrieb verdeutlicht. Wird die Karte an ein Lesegerät gehalten, wird der Strom zum Betrieb der Karte induziert und die Karte damit aktiviert. Anschließend erfolgt die Auswahl der vom Lesegerät gewünschten Applikation, wie z.B. der FIDO2-Anwendung, und die erste Befehls-APDU wird an die Karte gesendet (1). Diese wird vom MCU entgegengenommen und zwischengepuffert. Bevor die Abarbeitung des Befehls durchgeführt wird, muss erst die Verifizierung des Fingerabdrucks vorgenommen und ein Kommunikationskanal zum SE aufgebaut werden. Dazu aktiviert

der MCU den Fingerabdrucksensor (2), welcher nun den Benutzer auffordert, seinen Finger zum Abgleich gegen das zuvor etablierte Template auf den Sensor zu platzieren. Nachdem das aktuelle Template für den Abgleich aufgenommen wurde, wird dies über das MCU an das im SE befindliche Biometrie-Applet weitergeleitet (3). Das Applet führt den Abgleich durch und speichert das Ergebnis zwischen. Anschließend wird ein Kommunikationskanal zwischen dem sicheren Element und dem MCU etabliert (4). Daraufhin wird die Befehls-APDU an das vom Lesegerät angefragte Applet über den Kanal weitergeleitet, in diesem Fall an die FIDO2-Anwendung. Dieses arbeitet den entsprechenden Befehl ab und kann nun, sofern benötigt, auf das Ergebnis des Fingerabdruckvergleichs, über eine vom Biometrie-Applet bereitgestellte Schnittstelle, zugreifen (5). Das Ergebnis des Befehls wird anschließend über den Kommunikationskanal zurück an die MCU geleitet (4), welche die Antwort-APDU anschließend per NFC an das Lesegerät zurück überträgt (1). Wurde das Protokoll der Anwendung entsprechend abgearbeitet, kann die Karte vom Lesegerät entfernt werden. Daraufhin wird die Stromversorgung unterbrochen und alle Inhalte des Arbeitsspeichers gehen verloren.

Diese Architektur gewährleistet, dass unautorisierte Daten nicht direkt in das SE eingeleitet werden. Bevor APDUs das SE betreten dürfen, muss die Zustimmung durch den Nutzer gegeben werden, in dem der Fingerabdruck gescannt wird. Weiterhin wird durch die Implementierung des biometrischen Vergleichs innerhalb eines eigenen Applets ein Schutz der sensiblen biometrischen Referenzdaten realisiert. Da die Daten nur einmalig vorliegen müssen und anschließend über die Schnittstelle nur auf das Ergebnis zugegriffen werden kann, erhalten die auf der Karte installierten weiteren Anwendungen nur die für sie benötigte Information über das Ergebnis des Abgleichs. Weitere Daten über die biometrischen Merkmale können nicht eingesehen werden. Dadurch bieten die weiteren Applets keine Angriffsmöglichkeiten auf die hinterlegten biometrischen Merkmale. Zudem spart diese Lösung Speicherplatz, da die Referenzdaten nur einmalig innerhalb des biometrieverarbeitenden Applets vorliegen müssen. Da das verwendete SE nach CC EAL 6+ zertifiziert ist, kann hier von einem hohen Schutz der im SE gespeicherten Daten ausgegangen werden.

Trotz der Sicherheitsaspekte, die von der Karte etabliert werden, bietet der Fingerabdruck selbst eine Schwachstelle im Design. So könnte ein versierter Angreifer die Kommunikation zwischen Fingerabdrucksensor und MCU abhören und so die darüber übermittelten biometrischen Daten mitschneiden. Anschließend könnten die abgefangenen Daten wieder über den Datenbus eingespielt werden. So könnte ein Angreifer die Verwendung des Fingerabdrucksensors simulieren, um unrechtmäßigen Zugriff auf die Karte zu erhalten. Dies setzt jedoch den physischen Besitz der Karte voraus. Sind die Mittel für einen solchen Angriff nicht gegeben, stellt auch die Fälschung des Fingerabdrucks eine weitere Angriffsmöglichkeit dar. Da eine Lebenderkennung mit dem Sensor nicht möglich ist, wird zur Verhinderung solcher Angriffe eine Fake-Erkennung innerhalb der Fingerabdruckkarte implementiert. Diese soll gefälschte Fingerabdrücke erkennen und abweisen. Die Vorlage eines zweidimensionalen Bildes sollte daher in der Regel nicht ausreichen, um den Sensor zu überlisten. Es müssen aufwändigere Verfahren angewandt werden, um erfolgreiche Ergebnisse erzielen zu können. Wurde hier jedoch

eine Methode gefunden, könnte ein entsprechender Angriff gut skalieren. Weiterhin stellt auch die Kommunikation zwischen Lesegerät und Karte einen Angriffspunkt dar. Hier müssen die Anwendungen jedoch selbst entsprechende Schutzmaßnahmen, wie in Abschnitt 6.1.3 beschrieben, bereitstellen.

6.4 Designentscheidungen

Der folgende Abschnitt beschreibt die Implementierung des FIDO2-Applets sowie die während der Implementierung getroffenen Designentscheidungen. Dazu werden zuerst die Eigenschaften des Tokens genauer beschrieben. Anschließend wird auf die Implementierung des Applets eingegangen.

6.4.1 Eigenschaften des Tokens

Im Rahmen der Arbeit wurde die Implementierung eines transportablen FIDO2-Authentifikators mittels Java Card 3.0.2 vorgenommen, welcher die passwortlose Authentifizierung gegenüber Diensten per NFC ermöglicht. Dies wird über die biometrische Authentifizierung mittels eines Fingerabdrucks gegenüber dem Token erreicht. Neben der biometrischen Authentifikation des Benutzers wurde auch die Funktionlität des clientPins bereitgestellt. Das Vertrauen zwischen Server und Token wird über die *Basic Attestation* aufgebaut. Die Speicherung der dienstspezifischen privaten Schlüssel wird ausschließlich serverseitig innerhalb der *credentialID* vorgenommen, die Verschlüsselung erfolgt dabei mit AES-256. Einige der zuvor genannten Eigenschaften sind dabei durch die Form und die Möglichkeiten der in Abschnitt 6.3 beschriebenen Fingerabdruckkarte vorgegeben worden. Darunter fallen die Mobilität, die Kommunikation per NFC, die Verwendung des Fingerabdrucks als biometrisches Merkmal sowie die Nutzung von Java Card 3.0.2 zur Implementierung. Java Card 3.0 ist zudem nach CC EAL 4+ zertifiziert [79]. Alle weiteren Eigenschaften wurden während der Implementierung des Tokens festgelegt.

Es wurde sich für eine serverseitige Speicherung der *credentialIDs* entschieden, da die Fingerabdruckkarte innerhalb des sicheren Elements nur 128 KByte EEPROM zum Speichern von Programmen und weiteren Daten besitzt. Neben dem FIDO2-Applet sollen auch weitere Anwendungen auf der Karte installiert werden, z.B. das Applet zum Abgleich des Fingerabdrucks. Weiterhin wird der Speicherplatz für andere Daten, wie beispielsweise das Template des Fingerabdrucks benötigt. Somit steht nur wenig Speicherplatz für die eigentliche Implementierung des FIDO2-Applets auf dem SE zur Verfügung. Theoretisch wäre ebenfalls eine Speicherung der *credentialIDs* innerhalb des MCUs möglich gewesen. Da dieser jedoch keinen besonderen Schutz der *credentialIDs* geboten hätte und auch hier der Speicher eine begrenzte Ressource darstellt, wurde sich ebenfalls gegen diese Lösung entschieden. Der Authentifikator soll möglichst zustandslos sein und daher mit einer unbegrenzten Anzahl von Diensten funktionieren. Speicherlimits hätten die Anzahl der mit dem Token verwendbaren Dienste nach oben hin beschränkt.

Um eine sichere Übertragung und Speicherung der *credentialID* zu gewährleisten, wurde die *credentialID* mittels eines während der Herstellung des Tokens zufällig generierten AES-Schlüssels mit AES-256 verschlüsselt. Zudem wird bei jeder Erstellung einer *credentialID* ein zufälliger Initialisierungsvektor für die aktuelle Verschlüsselung generiert. Dieser wird nach der Verschlüsselung der *credentialID* an diese angehängt. Dieses Vorgehen schützt somit den dienstspezifischen privaten Schlüssel während der Übertragung und Speicherung auf dem Server. Zudem wird dadurch gewährleistet, dass nur der originale Authentifikator die *credentialID* korrekt entschlüsseln kann. AES-256 war dabei – soweit bekannt – das stärkste von der Karte bereitgestellte symmetrische Verfahren. Bei der Wahl der Algorithmen wurde zudem auf die Konformität zur FIDO Authenticator Allowed Cryptography List [70] geachtet.

Zur Generierung des dienstspezifischen Schlüsselpaares während der Registrierung des Authentifikators wird die elliptische Kurve *NIST P-256* (auch *Secp256r1*) benutzt. Neben dieser Kurve wurden von der FIDO Alliance auch die *NIST P-384* (auch *Secp384r1*) sowie die *NIST P-521* (auch *Secp521r1*) zur Benutzung spezifiziert [80]. Die maximale unterstützte Schlüssellänge für elliptische Kurven durch die Karte lag jedoch bei 256 Bit, weshalb die Wahl entsprechend ausfiel.

Alternativ hätte anstellte des dienstspezifischen privaten Schlüssels auch kryptographisches Material innerhalb der *credentialID* gespeichert werden können. Dieses kryptographische Material könnte anschließend für eine deterministische Schlüsselableitung bei jeder Authentifizierung gegenüber einem Dienst verwendet werden. So müsste nicht der entsprechende dienstspezifische private Schlüssel nach außen gegeben werden. Würde ein Angreifer an das kryptographische Material kommen, müsste er zudem zusätzlich den zur Schlüsselableitung verwendeten privaten Schlüssel aus dem Authentifikator extrahieren, um einen erfolgreichen Angriff durchzuführen. Aufgrund des prototypischen Charakters und der Einfachheit der zuvor genannten Lösung, wurde sich gegen das hier beschriebene Vorgehen entschieden. Eine Anwendung der deterministischen Schlüsselableitung in einer späteren Implementierung wäre denkbar.

Die Entscheidung zur passwortlosen Authentifizierung basiert dabei auf der Designentscheidung der Auslagerung der *credentialIDs*. Für die benutzerdatenlose Authentifikation wäre die Speicherung der *credentialIDs* auf dem Token notwendig gewesen. Da bei einer benutzerdatenlosen Anmeldung der Server nicht wissen kann, welcher Benutzer sich anmeldet, kann der Server nur die *ServerID* bereitstellen. Anschließend muss das Token selbst, anhand der vom Clienten erhaltenen Daten, die zum Dienst dazugehörige *credentialID* innerhalb seiner internen Speicherstruktur finden. Erst aus der *credentialID* geht hervor, welcher Benutzer bei dem Dienst angemeldet werden soll. Im Anschluss muss der Authentifikator diese Information an den Server übermitteln [4]. Da diese Möglichkeit bei der serverseitigen Speicherung nicht gegeben ist, konnte die benutzerdatenlose Authentifizierung nicht umgesetzt werden. Zudem erfordert die Verwendung des passwortlosen Ansatzes keine regelmäßige Übermittlung der *Benutzerinformationen* an den Server.

Zusätzlich zur biometrischen Authentifikation wurde der *clientPin*, wie in Kapitel 4.1.2 beschrieben, bereitgestellt. Auf diese Weise hat der Benutzer neben der biometrischen Authentifizierung eine weitere Authentifikationsmöglichkeit gegenüber dem Token. Dies

ermöglicht ihm die Wahl des anzuwendenden Verfahrens. Zudem wird eine Alternative geboten, falls der Fingerabdruck aufgrund einer Verletzung nicht verwendet werden kann. Dabei wäre eine Kombination beider Verfahren ebenfalls möglich, welche derzeit jedoch nicht implementiert ist. Neben der Auswahlmöglichkeit für den Benutzer erfolgte die Implementierung der *clientPin* ebenfalls zur vollständigen Unterstützung der in Kapitel 4.2.1 vorgestellten FIDO2-API für Authentifikatoren.

Als Vertrauensmodell (siehe Kapitel 4.1.3) wurde die *Basic Attestation* gewählt. Aufgrund der Einfachheit des Systems und der schnelleren Testbarkeit des Tokens im Vergleich zur *Attestation CA* und der *ECDAA* sowie des stärkeren Vertrauensbeweises gegenüber der *Self Attestation* wurde sich für dieses Verfahren entschieden. Dabei zeichnet sich die Einfachheit dadurch aus, dass keine dritte Partei in den Prozess mit eingebunden werden muss, welche für die Zertifikatsausstellung zuständig ist. Für die *Basic Attestation* wurde ein für die „Geräteserie“ selbst ausgestelltes serienspezifisches Zertifikat und das dazugehörige selbst generierte serienspezifische Schlüsselpaar verwendet. Die Generierung des Schlüsselpaares wurde dabei ebenfalls mit der elliptische Kurve *NIST P-256* durchgeführt.

Da das Ziel der Arbeit die Implementierung eines FIDO2-Tokens zur passwortlosen Authentifizierung gegenüber verschiedenen Diensten war, wurde zudem nur das CTAP2-Protokoll implementiert. Eine Unterstützung des CTAP1 / U2F-Protokolls könnte jedoch später hinzugefügt werden, um so ebenfalls die 2FA zu ermöglichen. Abschließend sei darauf hingewiesen, dass die maximale durch den Authentifikator unterstützte Nachrichtenlänge 1024 Bytes entspricht. Dies ist ebenfalls auf den begrenzten Speicher der Smartcard und die Vorgaben durch den Standard [5] zurückzuführen.

6.4.2 Implementierung

Die Implementierung der FIDO2-Anwendung erfolgte mit Java Card in der Version 3.0.2. Dabei wurde die in Kapitel 4.2.1 beschriebene Schnittstelle für Authentifikatoren vollständig implementiert. Zusätzlich mussten weitere Komponenten entwickelt werden, wie ein CBOR De- und Encoder, der die von der FIDO Alliance in [5] definierten Anforderungen zum (De-)Kodieren von Nachrichten erfüllte. Weiterhin wurde eine Routine zum einmaligen Aufspielen des gerätespezifischen privaten Schlüssels sowie des dazugehörigen Zertifikats bereitgestellt.

Um sicherzustellen, dass der Authentifikator nur dann eingesetzt werden kann, wenn der Herstellungsprozess vollständig abgeschlossen ist, wurde eine Zustandsmaschine implementiert. Diese Zustandsmaschine bildet zudem den aktuellen Zustand des Authentifikators während der Operationsdurchführung ab. In Abbildung 12 werden dabei die einzelnen Zustände des Authentifikators sowie die Zustandsübergänge dargestellt. Aus der Grafik geht hervor, dass, nach der Installation des Applets auf der Fingerabdruckkarte, der Status „Initialisierung“ angenommen wird. Dieser kann erst verlassen werden, wenn das geräteserienspezifische Zertifikat sowie der dazugehörige Schlüssel auf der Karte hinterlegt wurden. Anschließend befindet sich das Token im Zustand „Bereit“. Wird eine Nachricht empfangen, wird unterschieden, welches Datenformat die eingehende APDU aufweist. Ist diese im Extended Length Format, kann die angefragte

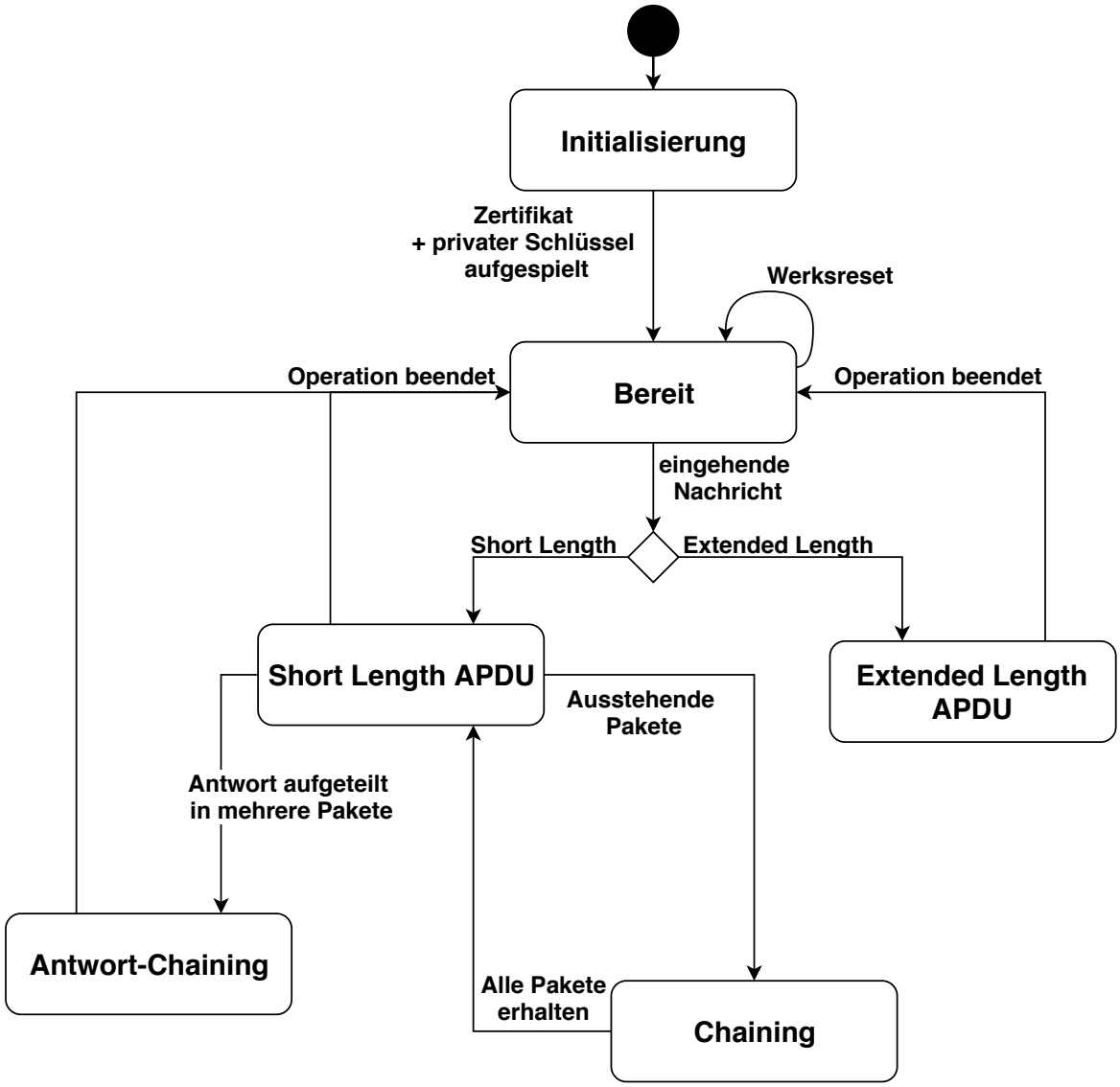

Abbildung 12: Implementierte Zustandsmaschine zur Darstellung des internen Authentifikatorzustands.

Operation ausgeführt und die Antwort des Tokens an das Lesegerät und somit den Clienten zurückgegeben werden. Daraufhin befindet sich das Token wieder im Zustand „Bereit“. Wird jedoch die Nachricht im Short Length Format an das Token gesendet, so kann es vorkommen, dass mehrere Short Length APDUs zum Übermitteln der kompletten Nachricht benötigt werden. Dies ist darauf zurückzuführen, dass das Token Nachrichten der Länge 1024 Bytes empfangen kann, eine Short Length APDU jedoch nur 256 Bytes an Daten übermitteln kann. In diesem Fall wird der Zustand „Chaining“ solange angenommen, bis der komplette Befehl beim Token eingegangen ist. Anschließend wird die Operation bearbeitet. Überschreitet auch hier die Antwortnachricht eine Länge von 256 Bytes, muss diese entsprechend in mehrere Short Length APDUs aufgeteilt

werden, um die Übermittlung des Ergebnisses zu ermöglichen. Dazu wird der Zustand „Antwort-Chaining“ eingenommen. Dies ist notwendig, da das eingehende Nachrichtenformat das Ausgabeformat des Tokens bestimmt. Wurde das Ergebnis komplett durch das Token übertragen, befindet sich der Authentifikator wieder im Zustand „Bereit“. Bei Ausführung eines Werksresets, wird der Zustand „Bereit“ beibehalten, Signaturzähler und eventuell vergebene PIN werden jedoch zurückgesetzt.

Zudem wurde bei der Implementierung darauf geachtet, dass Nachrichtenlänge sowie die enthaltenen Daten auf Vollständigkeit und Struktur geprüft werden. Zur Überprüfung der Struktur und der erwarteten Inhalte konnte der im Rahmen dieser Arbeit entwickelte CBOR-Decoder verwendet werden. Dieser stellt entsprechende Funktionalitäten zum Testen von Nachrichteninhalten bereit. Nur wenn alle benötigten Parameter vorhanden sind und die Struktur dem Standard entspricht, werden die angefragten Operationen durch das Token ausgeführt. Die zur Implementierung der Operationen verwendeten kyrptographischen Verfahren wurden dabei auf Basis der FIDO Authenticator Allowed Cryptography List [70] gewählt.

Aufgrund der in Kapitel 6.1.1 genannten Einschränkungen von Java Card, mussten während der Entwicklung zudem einige alternative Lösungen gefunden werden. Beispielsweise war es nicht möglich, normale Zeichenketten zu verwenden. Stattdessen mussten Abgleiche von Textinhalten in der byteweisen Repräsentation als *char* erfolgen. Auch die Implementierung des Signaturzählers [5] erforderte die Verwendung von zwei Zahlen des Typs *short*, bei denen eine Überlaufbehandlung während der Addition durchgeführt werden muss. Da der Wert nur in der binären Repräsentation innerhalb des Tokens relevant ist, stellt dieses Vorgehen jedoch kein Problem dar.

Abschließend wurde zum Ende der Implementierung festgestellt, dass eine Optimierung der Speicherverwendung innerhalb des Tokens notwendig ist. Da sehr viele Operationen auf Daten im EEPROM zurückgreifen und Ergebnisse dort zwischenspeichern, leidet die Performance der FIDO2-Anwendung unter den Speicherzugriffen.

7 Ergebnisse

Im Rahmen dieser Arbeit konnte gezeigt werden, dass ein funktionsfähiger FIDO2-Authentifikator zur passwortlosen Authentifizierung in Form einer Smartcard mit biometrischem Sensor entwickelt werden konnte, welcher ausschließlich durch das elektromagnetische Feld des NFC-Lesegerätes mit Strom versorgt wird. Im Folgenden werden die Ergebnisse der Untersuchung der Zertifizierbarkeit des Tokens dargestellt. Neben der biometrischen Zertifizierbarkeit wird dabei auch die funktionale Zertifizierbarkeit betrachtet. Diese umfasst die während der Entwicklung durchgeführten Selbsttests sowie die Ergebnisse der Tests gegenüber verschieden Referenzdiensten. Abschließend wurden die Anforderungen [68] an einen Authentifikator des Levels 3+ als Grundlage der Authentifiktorzertifizierung betrachtet und die Implementierung auf Basis dieser Anforderung bewertet.

7.1 Biometrische Zertifizierbarkeit

Da ein biometrischer Sensor zum Abgleich des Fingerabdrucks verwendet wird, benötigt dieser eine biometrische Zertifizierung der FIDO Alliance. Dieser Abschnitt betrachtet dabei kurz die Anforderungen in Version 1.1 [67] der FIDO Alliance an biometrische Sensoren und gibt eine Einschätzung über die mögliche Zertifizierbarkeit des Sensors.

Die biometrische Zertifizierung [20] wird dabei separat für die biometrische Komponente durchgeführt. Dazu stellt der Hersteller die zum Test benötigte Hard- und Software dem FIDO-akkreditierten Sicherheitslabor, welches die Zertifizierung durchführt, bereit. Die FIDO Biometrics Requirements [67] beschreiben dabei die Anforderungen, welche ein Sensor für die Zertifizierung bereitstellen muss. So wird der Sensor auf den Schutz gegen „Presentation Attacks“ untersucht. Diese werden dazu in die drei Level A, B und C eingeteilt. Weitere Anforderungen werden zudem an die FAR und FRR des Sensors gestellt. Durch den Sensor muss dabei eine FAR von unter $1 : 10000$ und eine FRR von unter $3 : 100$, jeweils bei einer oberen Schranke des Konfidenzintervalls von 80%, erreicht werden. Nachfolgend wird ein kurzer Überblick über die Angriffs-Level gegeben.

Level A Level A umfasst Angriffe, bei denen der Fingerabdruck auf eine 2D-Fläche, wie zum Beispiel Papier, gedruckt wird und auf den Fingerabdrucksensor gelegt wird. Während des Tests werden sechs Angriffe dieses Levels gegen das Token getestet. Für fünf der sechs Tests sollte dabei die Erfolgsrate des jeweiligen Angriffs unter 20% liegen. Insgesamt sollte die Erfolgsrate aller ausgeführten Angriffe dieses Levels unter 50% fallen.

Level B Level B umfasst fortgeschrittenere Angriffe, bei denen das Abbild des Fingerabdrucks auf eine Gussform übertragen wird und mittels verschiedener Materialien ausgefüllt wird. Dies wäre beispielsweise das Ätzen des Fingerabdrucks auf eine Leiterplatte, welche anschließend mit einem Material ausgefüllt und z. B. mit Graphit bestäubt wird. Anschließend wird der so vorbereitete Abdruck gegen den Sensor geprüft. Während der Zertifizierung werden vier verschiedene Level B-Angriffe gegen das Token getestet. Dabei sollte die Erfolgsrate bei drei der vier Angriffen jeweils geringer als 20% sein. Wie bei Level A sollte zudem die Erfolgsrate aller ausgeführten Angriffe des Levels B unter 50% liegen.

Level C Die unter Level C gelisteten Angriffe sind sehr fortgeschrittene Techniken, bei denen ein 3D-Druck des Fingerabdrucks vorgenommen wird oder ein Model, welches zusätzlich die Venen, das Schwitzen oder den Blutfluss des Fingers simuliert, verwendet wird. Eine Angabe zu der Verwendung von Angriffen des Levels C während der Zertifizierung wurde dabei nicht vorgenommen.

Wie aus Kapitel 6.3 jedoch hervorgeht, wird durch die biometrische Lösung der Fingerabdruckkarte ein FAR von $1 : 1000$ und eine FRR von $2 : 100$ erreicht. Somit erfüllt die FAR die Anforderung von $1 : 10000$ nicht. Neben den in dem Kapitel angegebenen Fehlerraten geht hervor, dass – in der Regel – der Schutz gegen „Presentation

Attacks“ des Levels A durch den Sensor gegeben sein sollte. Somit werden Maßnahmen des Levels B erforderlich, um einen erfolgreichen Angriff durchzuführen. Da auch hier keine entsprechenden Analysen vorliegen, kann jedoch keine Konformität zum Standard ermittelt werden.

Um eine genaue Analyse der biometrischen Zertifizierbarkeit durchführen zu können, müsste zunächst eine empirische Analyse der FAR, der FRR sowie der Angriffe erfolgen. Aufgrund der vorliegenden Daten wäre jedoch eine Zertifizierung des biometrischen Sensors zum jetzigen Zeitpunkt nicht erfolgreich.

7.2 Funktionale Zertifizierbarkeit

Die funktionale Zertifizierbarkeit [19] ist die Voraussetzung für die Authentifikatorzertifizierbarkeit. Während der Entwicklung wurden dabei verschiedene Tests bzgl. der Funktionalität des Tokens angestellt. Im Folgenden werden der durchgeführte Selbsttest sowie die durchgeführten Interoperabilitätstests genauer beschrieben. Abschließend wird eine Einschätzung der funktionalen Zertifizierbarkeit auf Grundlage der Richtlinien in Version 1.3.7 gegeben.

7.2.1 Selbsttest

Während der Entwicklung erfolgte ein Selbsttest der vom Token generierten Antworten auf eingehende Registrierungs- und Authentifizierungsanfragen. Da kein Zugriff zum Testtool der FIDO Alliance gegeben war, wurde dafür die von Yubico entwickelte und durch Frank Morgner für die Kommunikation mit APDUs angepasste *libfido2*[5] verwendet. Für die Durchführung des Tests wurde das in Implementierung befindliche Applet mittels *jCardSim*[6] simuliert. Die Kommunikation zwischen *jCardSim* und *libfido2* wurde mittels des *BixVReaders*[7] ermöglicht.

Die Grundfunktionalität der *libfido2* besteht darin, Registrierungs- und Authentifikationsnachrichten an den zu testenden Authentifikator zu schicken und die Struktur sowie den Inhalt der generierten Antworten auf Konformität gegenüber dem Standard zu verifizieren. Zudem kann ein Werksreset durchgeführt sowie der *clientPin* getestet werden. Einer der Vorteile des Einsatzes der Bibliothek war die gegebene Möglichkeit des Debuggings des Applets während der Befehlsausführung. Zudem konnte schon zu Beginn der Entwicklung die bereits implementierte Funktionalität des Tokens verifiziert werden. Nachdem alle Tests zur Registrierung und Authentifizierung gegenüber der Bibliothek bestanden wurden, wurde die *clientPin*-Funktionalität implementiert und die ersten Tests auf Interoperabilität mit verschiedenen Diensten durchgeführt. Jedoch stellte sich während des Interoperabilitätstests heraus, dass der Inhalt der Antworten durch die *libfido2* nicht streng auf Konformität überprüft wurde, sodass auch nicht dem Standard entsprechende Nachrichten als korrekt angesehen wurden.

[5] `https://github.com/frankmorgner/libfido2` (Stand: 15.11.2019).
[6] `https://jcardsim.org/` (Stand: 15.11.2019).
[7] `https://github.com/frankmorgner/vsmartcard/tree/master/virtualsmartcard/win32/BixVReader` (Stand: 15.11.2019).

Da während der Entwicklung kein Zugriff auf das entsprechende Tool bestand, wurde dieses alternative Vorgehen gewählt. Dies entsprach jedoch nicht dem Vorgehen der funktionalen Zertifizierung, da nach einem erfolgreichen Selbsttest keine Änderungen am Token mehr vorgenommen werden dürfen. Es wurden jedoch erst nach erfolgreichem Test des FIDO2-Applets die Installation auf der Fingerabdruckkarte vorgenommen und die Benutzerauthentifikationsmethoden implementiert. Weiterhin ist unklar, ob der durch *libfido2* durchgeführte Test vollständig dem des Testtools entspricht. Aus diesen Gründen wurde daher kein Selbsttest nach FIDO-Kriterien durchgeführt, wie er aus dem Dokument zur funktionalen Zertifizierung [19] hervorgeht. Um die funktionale Zertifizierung des Tokens erfolgreich durchführen zu können, muss zuerst der im dazugehörigen Dokument beschriebene Selbsttest mittels des Testtools durchgeführt werden. Eine Einschätzung des Ergebnisses kann daher nicht vorgenommen werden.

7.2.2 Interoperabilitätstest

Der Interoperabilitätstest [19] wurde sowohl während der Entwicklung als auch nach der Fertigstellung des FIDO2-Applets mit der Fingerabdruckkarte durchgeführt. Als Referenz stand der FIDO2-zertifizierte YubiKey 5 NFC[8] (im Folgenden: YubiKey) von Yubico zur Verfügung. Der YubiKey unterstützt dabei die passwort- und benutzerdatenlose Authentifizierung mittels FIDO2 sowie die 2FA mittels U2F. Die Schlüssel wurden dabei unter Microsoft Windows 10 mit Windows Hello und auf dem Samsung Galaxy S10 unter Android 9 getestet. Für die Tests wurden fünf verschiedene Webdienste, welche speziell zum Testen von FIDO2-Authentifikatoren entwickelt wurden, verwendet. Testversuche mit Mozilla Firefox und Microsoft Edge führten häufig zu Problemen aufgrund von JavaScript-Inkompatibilitäten, daher wurde für den Test nur Google Chrome in der Version 76.0 benutzt. Bei den Tests wurden, je nach Möglichkeiten der Dienste, wie in [19] gefordert, die Registrierung und die Authentifizierung mittels *clientPin* bzw. Fingerabdruck sowie die Multi-Account-Anwendung getestet. Die zur Zertifizierung vorgesehene Überprüfung des Umgangs mit unbekannten Erweiterungen des FIDO2-Protokolls wurde dabei, soweit bekannt, durch keine der Testseiten unterstützt.

Im Folgenden werden die Ergebnisse der Tests der einzelnen Komponenten erläutert.

Client 1: Windows Hello

Die Verwendung der Fingerabdruckkarte mit Windows Hello funktionierte für alle Anwendungsfälle. So konnte sowohl die Registrierung als auch die Authentifizierung mittels der Fingerabdruckkarte sowohl mit *clientPin* als auch mit Fingerabdruck durchgeführt werden. Auch das Enrollment sowie die Verwaltung der PIN waren mittels Windows Hello möglich. Bei Verwendung des *clientPins* wurden jedoch Fehler im Umgang mit dem Fehlversuchszähler durch Windows Hello festgestellt. Obwohl nur drei aufeinanderfolgende Falscheingaben zulässig sein sollten, bevor die aktuell angefragte Operation neu gestartet werden muss,

[8] `https://support.yubico.com/support/solutions/articles/15000014174--yubikey-5-nfc` (Stand: 15.11.2019).

wurde die vom Token nach drei aufeinanderfolgenden Falscheingaben generierte Fehlermeldung ignoriert und weitere PIN-Eingaben durch Windows Hello zugelassen. Das maximale Limit von acht PIN-Eingaben bis zur Sperrung des Tokens wird jedoch beachtet [5]. Gleiche Ergebnisse wurden für den YubiKey erzielt.

Client 2: Android 9

Die Verwendung der Fingerabdruckkarte als FIDO2-Token war unter Android 9 nicht möglich. Die Karte wurde zwar per NFC erkannt, jedoch wurde keine Interaktion mit der Karte gestartet. Dies gilt sowohl bei der Verwendung mit *clientPin* als auch bei der Verwendung mit Fingerabdruck. Das gleiche Ergebnis wurde bei der Verwendung des YubiKeys festgestellt. Eine Verwendung des YubiKeys zur 2FA mit den nachfolgend vorgestellten Diensten war, mit Ausnahme von Dienst 5, jedoch möglich.

Dienst 1: `https://webauthn.io/`[9]

Die Registrierung und Authentifizierung mittels der Fingerabdruckkarte erfolgte bei Verwendung des biometrischen Sensors fehlerfrei. Wurde jedoch die *clientPin*-Funktionalität zur Authentifizierung gegenüber dem Token angewendet, traten Probleme in der Nutzung auf. Wurde noch keine *clientPin* durch den Benutzer vergeben, so brach die auszuführende Operation nach gegebenem Timeout ab. Ein Enrollment der PIN war dabei nicht möglich. War die PIN jedoch bereits durch den Benutzer gesetzt, so konnte die Registrierung des Tokens gegenüber der Seite durchgeführt werden. Auch eine Änderung der PIN war möglich. Ein auf die Registrierung folgender Login mittels der *clientPin* war trotz erfolgreicher Registrierung jedoch nicht möglich. Weiterhin war die Nutzung des Tokens für mehrere bei dem Dienst angemeldete Accounts möglich. Gleiche Ergebnisse wurden mit dem YubiKey erzielt.

Dienst 2: `https://webauthn.bin.coffee/`[9]

Die Registrierung und Authentifizierung mittels Fingerabdruckkarte erfolgte sowohl bei Verwendung des Fingerabdrucks als auch bei Verwendung der *clientPin* erfolgreich. Ein Enrollment der *clientPin* war bei Bedarf ebenfalls möglich. Auch die Änderung der PIN konnte durchgeführt werden. Da die Seite nur einen Account zum Testen des Tokens bereitstellte, war der Multi-Account-Test nicht möglich. Gleiche Ergebnisse wurden mit dem YubiKey erzielt.

Dienst 3: `https://webauthn.me/`[9]

Die Registrierung und Authentifizierung mittels Fingerabdruckkarte erfolgte auch bei diesem Dienst sowohl bei Verwendung des Fingerabdrucks als auch bei Verwendung der *clientPin* erfolgreich. Ein Enrollment der *clientPin* war bei Bedarf ebenfalls möglich. Auch die Änderung der PIN konnte durchgeführt werden. Die Verwendung des Tokens mit mehreren Accounts war ebenfalls erfolgreich. Gleiche Ergebnisse wurden mit dem YubiKey erzielt.

[9]Stand: 15.11.2019.

Dienst 4: `https://demo.yubico.com/`[10]

Die Registrierung und Authentifizierung mittels der Fingerabdruckkarte erfolgte bei Verwendung des biometrischen Sensors fehlerfrei. Wurde jedoch die *clientPin*-Funktionalität zur Authentifizierung gegenüber dem Token angewendet, traten Probleme in der Nutzung auf. Wurde noch keine *clientPin* durch den Benutzer vergeben, so brach die auszuführende Operation nach gegebenem Timeout ab. Ein Enrollment der PIN war dabei nicht möglich. War die PIN jedoch bereits durch den Benutzer gesetzt, so konnte die Registrierung des Tokens gegenüber der Seite durchgeführt werden. Auch eine Änderung der PIN war möglich. Ein auf die Registrierung folgender Login mittels der *clientPin* war trotz erfolgreicher Registrierung jedoch nicht möglich. Da die Seite nur einen Account zum Testen des Tokens bereitstellte, war der Multi-Account-Test nicht möglich. Die Probleme hinsichtlich der *clientPin* konnten bei der Verwendung des YubiKeys nicht festgestellt werden. Der YubiKey konnte, mit Ausnahme der Multi-Account-Anwendung, alle Tests erfolgreich abschließen.

Dienst 5: `https://webauthn.spomky-labs.com/`[10]

Die Funktionalität dieses Dienstes war browserübergreifend unzuverlässig, was auf JavaScript-Inkompatibilitäten zurückzuführen war. Dies führte dazu, dass nur wenige Testversuche mit der Fingerabdruckkarte erfolgreich durchgeführt werden konnten. Häufig führten die Fehler der Dienste zum Abbruch der ausgeführten Operation. Die erfolgreichen Tests wurden für die Registrierung und Authentifizierung mit dem Fingerabdruck getestet. Diese waren ohne Probleme möglich. Auch der Multi-Account-Test war erfolgreich. Aufgrund der unzuverlässigen Funktionalität der Seite konnten keine Daten mit dem YubiKey erhoben werden.

Trotz der unterschiedlichen Ergebnisse der Tests, konnten alle im Rahmen der Arbeit implementierten Funktionalitäten des Authentifikators getestet werden. Da der YubiKey – mit Ausnahme von Dienst 4 und 5 – die gleichen Ergebnisse erzielte, ist davon auszugehen, dass die festgestellten Probleme nicht nur auf den in dieser Arbeit entwickelten Authentifikator zurückzuführen sind. Auch die Implementierungen der Testdienste scheinen Fehler und Inkompatibilitäten aufzuweisen. Ein Beispiel dafür sind die durch JavaScript-Inkompatibilitäten entstandenen Probleme. Dennoch sind durch die Testergebnisse die Anforderungen des Interoperabilitätstests [19] nicht erfüllt. Hierzu müssen alle Funktionalitäten des Authentifikators mit allen Testdiensten fehlerfrei funktionieren.

7.2.3 Erkenntnisse

Da sowohl der Selbsttest, als auch der Interoperabilitätstest nicht in allen erforderlichen Punkten erfolgreich waren, ist derzeit keine funktionale Zertifizierbarkeit gegeben [19].

[10] Stand: 15.11.2019.

Jedoch konnte durch die Tests gezeigt werden, dass alle implementierten Funktionalitäten funktionieren und die auftretenden Probleme nicht ausschließlich auf den entwickelten Authentifikator zurückzuführen sind. Dennoch sollte hier ein geeignetes Testumfeld gefunden werden, bei dem die Funktionalität von Seiten der Dienste zuverlässig erbracht wird. Nur dann können gegebenenfalls noch immer in der Implementierung vorhandene Fehler gefunden und beseitigt werden.

Durch die Tests wurden zudem einige Fehler und Unzulänglichkeiten in den Beschreibungen des FIDO2-Standards [5, 4] sowie bei den Diensten und Clienten festgestellt. Beispielsweise weist die in [5] spezifizierte Antwort für **authenticatorMakeCredential()** eine andere Reihenfolge in der Kodierung der Struktur auf. Anstelle der spezifizierten Struktur *authenticatorData - format - attestationStatement* wird die Struktur *format - authenticatorData - attestationStatement* von den Diensten verwendet.

Weiterhin wurde festgestellt, dass, wenn mehrere Tokens für einen Account hinterlegt werden und das aktuell verwendete Token die Speicherung der *credentialIDs* auf den Server auslagert, alle für den Account hinterlegten *credentialIDs* einzeln an den verwendeten Authentifikator übermittelt werden. Dieser prüft die aktuell übermittelte *credentialID* und gibt, sofern nicht bekannt, einen Fehler zurück. Daraufhin wird die nächste *credentialID* an das Token gesendet. Sobald die vom Token generierte *credentialID* an das Token übergeben wird, erfolgt die Generierung der Signatur und die Authentifizierung gegenüber dem Dienst ist erfolgreich.

Zudem wurden auch Probleme bzgl. der Anwendung der *clientPin* und der Fehlversuchszähler gefunden. Während nur Dienst 2 und Dienst 3 eine vollständige und fehlerfreie Unterstützung der *clientPin* anboten, ignorierte Windows Hello die vom Token geworfenen Fehlermeldungen bei zu vielen direkt aufeinanderfolgenden Fehleingaben. Diese Fehlermeldungen sollten einen Neustart der aktuell ausgeführten Operation erzwingen. Stattdessen ermöglichte Windows Hello weitere PIN-Eingaben. Das Limit von maximal acht falschen PIN-Eingaben vor Sperrung des Tokens wurde jedoch durch Windows Hello beachtet [5]. Weiter konnte festgestellt werden, dass die FIDO2-Funktionalität des passwortlosen und benutzerdatenlosen Logins unter Android 9 zum Zeitpunkt der Tests im Juli/August 2019 nicht unterstützt wurde.

7.3 Authentifikatorzertifizierbarkeit

Die Betrachtung der Zertifizierbarkeit des Authentifikators wurde auf Grundlage der in den FIDO Authenticator Security Requirements in Version 1.3.0 [68] definierten Anforderungen durchgeführt. Dabei wurde die Zertifizierung auf Level 3+ betrachtet. Von den 83 in dem Dokument enthaltenen Anforderungen sind derzeit 80 Anforderungen für FIDO2-Authentifikatoren zu evaluieren. Eine wegfallende Anforderung bezieht sich dabei ausschließlich auf UAF-Authentifikatoren. Die anderen beiden Anforderungen sind aufgrund von Überarbeitungen durch die FIDO Alliance von der Zertifizierung ausgeschlossen worden.

Innerhalb des Anforderungsdokuments [68] werden die Anforderungen in neun Kategorien eingeteilt. In der nachfolgenden Auflistung werden die Inhalte der neun Kategorien kurz beschrieben.

Kategorie 1: Allgemeine Beschreibung des Authentifikators

Die erste Kategorie umfasst die Definition des Umfangs des zu zertifizierenden Authentifikators. Dazu gehören die verwendeten kryptographischen Algorithmen, Ein- und Ausgabemöglichkeiten, die Beschreibung der Ausführungsumgebung sowie die Umsetzung einiger Softwareeigenschaften.

Kategorie 2: Schlüsselmanagement und Schutz der Sicherheitsparameter

Kategorie zwei umfasst das Schlüsselmanagement und den Schutz der verwendeten Schlüssel und geheimen Parameter innerhalb und außerhalb des Authentifikators. Zudem wird die Sicherheit der verwendeten kryptographischen Verfahren betrachtet, wozu auch der Zufallszahlengenerator mit einbezogen wird.

Kategorie 3: Benutzerauthentifizierung

Diese Kategorie stellt Anforderungen an die verwendeten Benutzerauthentifizierungverfahren bzw. den Test auf Anwesenheit eines Benutzers.

Kategorie 4: Privatsphäre

In Kategorie 4 werden Anforderungen zum Schutz der Privatsphäre gestellt.

Kategorie 5: Schutz vor Angriffen

Kategorie 5 befasst sich mit der Resistenz gegen physische Angriffe, Seitenkanalangriffe und Fehlerinjektionsmöglichkeiten.

Kategorie 6: Vertrauensmodell

Unter dieser Kategorie werden Anforderungen an das verwendete Vertrauensmodell des Authentifikators zusammengefasst.

Kategorie 7: Laufzeit- und Ausführungsumgebung

In dieser Kategorie werden Anforderungen an die Sicherheit der verwendeten Laufzeit- und Ausführungsumgebung des Authentifikators definiert.

Kategorie 8: Firmwareupdates und Self-Tests

Die in dieser Kategorie gestellten Anforderungen beziehen sich auf durchzuführende Selbst-Tests kryptographischer Funktionen durch den Authentifikator während des Betriebs im Feld. Zudem werden Anforderungen an die Durchführung von Firmwareupdates definiert.

Kategorie 9: Herstellungsprozess

In Kategorie 9 werden Anforderungen an den Herstellungsprozess gestellt. Diese umfassen u.a. die Versionierung der Entwicklungsdokumente und des Codes sowie den Zugriff auf sensible Informationen wie Gerätezertifikate oder geräteserienspezifische private Schlüssel.

Zur Durchführung der Zertifizierung sollen die Maßnahmen zur Erfüllung der in den verschiedenen Kategorien gestellten Anforderungen durch den Hersteller beschrieben werden. Diese Beschreibungen dienen anschließend der FIDO Alliance und/oder den Sicherheitslaboren als Grundlage zur Überprüfung des beantragten Zertifizierungslevels. Dabei wird durch die FIDO Alliance eine Excel-Tabelle mit den für das jeweilige Sicherheitslevel relevanten Anforderungen zur Unterstützung des Herstellers bereitgestellt [6]. Die in dieser Arbeit durchgeführte Analyse der Anforderungen wurde dabei mit Hilfe der Excel-Tabelle für Level 3+ durchgeführt[11]. Dabei wird für 61 der 80 gestellten Anforderungen nur eine Beschreibung der Umsetzung durch den Hersteller gefordert. Lediglich für 19 Anforderungen muss der Hersteller selbst eine Einstufung des erfüllten Sicherheitslevels durch die im Token umgesetzte Lösung geben. Diese umfassen Anforderungen aus allen Kategorien mit Ausnahme der Kategorie 6 [72].

Neben den 19 selbst einzustufenden Anforderungen werden die dazugehörigen Einstufungskriterien der unterschiedlichen Level für jede der Anforderungen durch die FIDO Alliance definiert [72]. Dabei können aufgrund der in Kapitel 6.3 vorgestellten Ausführungsumgebung sowie der verwendeten kryptographischen Algorithmen 13 von 19 Anforderungen auf Level 3+ eingestuft werden. Dies ist in der Regel darauf zurückzuführen, dass das verwendete SE nach CC EAL 6+, JCOP3 nach CC EAL 5+ und Java Card 3.0 nach CC EAL 4+ zertifiziert sind und die Implementierung weiterer auf der Karte ausgeführten Anwendungen nach CC-Kriterien erfolgte. Daher kann hier von einem guten Schutz für auf dem Authentifikator gespeicherte Daten, eine sichere Implementierung der kryptographischen Algorithmen sowie einer isolierten Ausführungsumgebung für das FIDO2-Applet ausgegangen werden. Zudem wurde sich während der Implementierung des FIDO2-Applets an die Vorgaben der FIDO Alliance gehalten und nur durch die FIDO Alliance erlaubte kryptographische Verfahren eingesetzt. Weiterhin werden alle an das FIDO2-Applet gesendeten Befehle auf das Vorhandensein aller benötigten und die Korrektheit aller übermittelten Parameter überprüft. Die Entwicklung erfolgte außerdem unter Anwendung eines Versionskontrollsystems, sodass alle Änderungen am Code nachzuvollziehen sind.

Weitere drei der 19 Anforderungen können derzeit nicht eindeutig bewertet werden, da das im Rahmen dieser Arbeit entwickelte Token ein Prototyp ist und daher die Anforderungen zum Herstellungsprozess [72] nicht beurteilt werden können. Dennoch ist davon auszugehen, dass die Anforderungen zum Herstellungsprozess in der späteren Produktion eingehalten werden können.

Somit können lediglich drei der 19 Anforderungen nicht durch das Token erfüllt werden. Zum einen ist dies auf die fehlende Zertifizierung des biometrischen Sensors zurückzuführen, welche eine Voraussetzung zur Authentifikatorzertifizierung eines biometrischen Tokens nach Level 3+ darstellt [20]. Zum anderen auf die Einbindung des biometrischen Sensors. Da dieser, wie in Kapitel 6.3 beschrieben, die aufgenommenen Daten unverschlüsselt an das sichere Element übermittelt, kann ein Angriff auf die Kommunikation des Fingerabdrucksensors ausgeführt werden. Durch die Anforderungen für Level 3+ wird jedoch eine sichere Übertragung der Daten gefordert, daher

[11] Die ausgefüllte Excel-Tabelle ist auf der im Anhang befindlichen CD-ROM verfügbar.

können diese Anforderungen durch die Fingerabdruckkarte nicht erfüllt werden. Da die Einstufungskritieren der FIDO Alliance für diese Anforderungen nur Level 3 und Level 3+ angeben, kann keine Einschätzung für andere Stufen erfolgen. Daher wird hier maximal von einem Erreichen der Stufe 2 ausgegangen [72].

Doch nicht nur die selbst einzustufenden Anforderungen wurden betrachtet. Für die weiteren 61 Anforderungen [72], die es zu überprüfen galt, wurden entsprechende Umsetzungsbeschreibungen vorgenommen. Auch hier sind einige Probleme aufgefallen, welche eine Zertifizierung auf ein höheres Level verhindern könnten. Da jedoch keine Einstufungskriterien für diese Anforderungen vorliegen, sondern diese durch die FIDO Alliance oder das Sicherheitslabor geprüft und beurteilt werden, kann hier keine Einschätzung des jeweils erreichten Levels vorgenommen werden. Einer der auffälligen Punkte ist hierbei die Unklarheit über die Einstufung des in der Implementierung genutzten Zufallszahlengenerators. Dabei handelt es sich um einen kryptographisch sicheren Zufallsgenerator, der von Java Card 3.0 bereitgestellt wird. Da der verwendete Algorithmus und die Einstufung jedoch nicht bekannt sind, ist unklar, ob dieser den in [70] genannten Anforderungen an einen Zufallsgenerator entspricht. Gleiches gilt für eventuell ermittelbare Laufzeitunterschiede der verwendeten kryptographischen Verfahren, welche nativ durch Java Card 3.0 bereitgestellt werden. Da Java Card 3.0 jedoch nach CC EAL 4+ zertifiziert ist [79], kann davon ausgegangen werden, dass entsprechende Anforderungen erfüllt sind. Weiterhin sind die zur Privatsphäre gestellten Anforderungen derzeit nicht erfüllt, da zurzeit nur das prototypische Token vorliegt. Da die *Basic Attestation* verwendet wurde, muss jedoch eine entsprechend große Geräteserie mit demselben Zertifikat ausgestattet werden, um später die Anonymität der Benutzer zu gewährleisten.

Weitere Probleme sind zudem in der Implementierung des FIDO2-Applets (siehe Kapitel 6.4) selbst zu finden. So wird der dienstspezifische private Schlüssel direkt in der *credentialID* herausgegeben. Diese ist zwar mit AES-256 verschlüsselt, ein Angreifer, der in den Besitz der *credentialID* kommt, kann jedoch einen Angriff auf die Verschlüsselung durchführen und auf diese Weise ggf. das zu schützende Geheimnis extrahieren. Zwar sollte mit AES-256 ein Schutz geboten werden, dennoch ist ein Risiko gegeben. Weiterhin ist aufgefallen, dass der Authentifikator keine Überprüfung der Übereinstimmung des Ursprungs der Nachricht mit dem in der *credentialID* hinterlegten Ursprung vornimmt. So kann ein Angreifer einen Man-in-the-Middle-Angriff durchführen, sofern der Client ebenfalls keine Überprüfung des Ursprungs der Daten vornimmt. Somit könnte der Angreifer in den Besitz einer gültigen Signatur kommen. Dies würde dem in Kapitel 2.2.3 beschriebenen Angriff entsprechen. Zudem sollte das Management der Schlüssel überdacht werden. Ein Werksreset des Tokens sollte z.B. dazu führen, dass der für die Verschlüsselung der *credentialID* verwendete Schlüssel erneut gewürfelt wird, damit frühere Registrierungen bei Diensten ungültig werden und auch die Freigabe von Operationen sollte nicht nur einmal pro Powerzyklus erfolgen. Hier müssen jedoch noch Anpassungen an der Software des MCU vorgenommen werden, der für die Ansteuerung des Fingerabdrucksensors zuständig ist.

Das Token benötigt zudem ein vom Hersteller bereitgestelltes Tool zum Enrollment des Fingerabdrucks. Dieses Tool könnte zudem für den Werksreset zuständig sein und

die Konfiguration des anzuwendenden Verfahrens zur Authentifizierung – *clientPin* oder Fingerabdrucksensor – ermöglichen. Auch die Möglichkeit eines Firmwareupdates des Tokens sollte gegeben werden. Die dafür notwendigen Anforderungen sind derzeit nicht erfüllt oder können nicht beurteilt werden, da ein entsprechendes Tool noch nicht vorliegt. Zudem müssen auch die Metadaten des Tokens für den FIDO-Metadaten-Service erstellt werden [72].

Aus der Analyse der Anforderungen geht hervor, dass derzeit keine Zertifizierung des Tokens nach Level 3(+) möglich ist, da der verwendete Fingerabdrucksensor und dessen Einbindung nicht den biometrischen Richtlinien [20] der FIDO Alliance entsprechen. Zudem kann nicht bei allen Anforderungen ein abschließendes Urteil gegeben werden, da nicht alle benötigten Informationen vorliegen. Auch die softwareseitige Implementierung des Authentifikators birgt Verbesserungspotential. Dies umfasst unter anderem den Schutz einiger Daten, die fehlende Möglichkeit für Firmwareupdates oder das Schlüsselmanagement. Dennoch konnte ebenfalls gezeigt werden, dass zum Teil Sicherheitsanforderungen [72] für Level 3+ erfüllt werden und durch geeignete Anpassung der Schwachstellen mindestens eine Zertifizierung nach Level 2 oder höher möglich sein könnte. Dies ist unter anderem auf das hohe Schutzniveau der Ausführungsumgebung – bestehend aus SE, JCOP3 und Java Card 3.0 – zurückzuführen (siehe Kapitel 6.3).

8 Fazit

Die Gedächtnisleistung des Menschen steht im Konflikt mit der stetig wachsenden Anzahl benötigter Passwörter. Daraus resultieren Sicherheitsrisiken für verwendete Benutzeraccounts, da aufgrund der hohen Anzahl an Passwörtern häufig schwache und einfach zu merkende gewählt werden. Um eine Entlastung der Gedächtnisleistung des Menschen zu erreichen und die Sicherheit der Benutzeraccounts zu gewährleisten, wurden viele verschiedene Ansätze, wie 2FA, Passwortmanager und Single Sign-On entwickelt. Dabei unterscheiden sich die verschiedenen Ansätze in Akzeptanz, Sicherheit und Verbreitung deutlich (siehe Kapitel 2). Die FIDO Alliance möchte daher mit ihren Standards eine einheitliche Schnittstelle für eine sichere Benutzerauthentifizierung gegenüber verschiedenen Diensten bereitstellen. Die Ziele der Standards sind neben der Vereinheitlichung der Prozesse auch die Sicherheit und Entlastung des Benutzers sowie die Wahrung der Privatsphäre der einzelnen Identitäten. Dazu werden sowohl verschiedene Schutzmechanismen in das Design der Protokolle mit einbezogen als auch die Datensparsamkeit der Protokolle beachtet. Dennoch konnten auch hier Schwachstellen gefunden werden (siehe Kapitel 2.2 und 4).

Im Rahmen dieser Arbeit wurde der neueste Standard der FIDO Alliance, der FIDO2-Standard, betrachtet. Dieser verspricht die passwort- oder benutzerdatenlose Authentifizierung gegenüber verschiedenen Webdiensten. Ziel der Arbeit war es, einen FIDO2-konformen Authentifikator zu entwickeln, der für die passwortlose Authentifikation bei Diensten mittels Fingerabdruck benutzt werden kann. Als Ausführungsumgebung wurde dabei die in Kapitel 6.3 beschriebene Fingerabdruckkarte vorgesehen. Eine Betrachtung der Sicherheit von Smartcards hat gezeigt, dass schon beim Design der

Komponenten die Sicherheit eine große Beachtung findet. Dennoch existieren je nach Sicherheit der Smartcard verschieden aufwändige Angriffe, durch welche die Sicherheitsmechanismen einer Smartcard unter Umständen erfolgreich umgangen werden können (siehe 6.1). Zudem wurde festgestellt, dass die Sicherheit der Biometrie zum Schutz von Daten und Zugängen von der Wahl des biometrischen Merkmals sowie von dem verwendeten Authentifizierungsverfahren abhängt. Dabei weist der Fingerabdruck unter Umständen Probleme in der quantitativen Erfassbarkeit des Merkmals auf. Auch die Fälschungssicherheit ist, je nach verwendetem Sensor und Algorithmus, unterschiedlich gut gegeben. Für einen Angriff muss dem potentiellen Angreifer jedoch erst einmal das entsprechende Merkmal, in diesem Fall der Fingerabdruck, vorliegen (siehe Kapitel 6.2).

Es konnte gezeigt werden, dass ein FIDO2-konformer Authentifikator auch unter herausfordernden Bedingung, wie z.B. dem Fehlen einer internen Batterie und wenig Speicherplatz, zu realisieren ist. Dabei wurde ein transportabler Authentifikator entwickelt, der passwortlose Authentifizierung mittels Fingerabdruck und FIDO2 über NFC ermöglicht. Die Stromversorgung erfolgt dabei ausschließlich durch das elektromagnetische Feld des NFC-Lesegerätes. Neben der Authentifizierung durch den Fingerabdruck wurde zusätzlich die Authentifizierung mittels *clientPin* ermöglicht (siehe Kapitel 6).

Während der Analyse der Zertifizierbarkeit des Tokens konnte die Funktionalität der FIDO2-Anwendung bestätigt werden. Eine definitive Aussage über die Zertifizierbarkeit konnte jedoch nicht gegeben werden. Dabei wurden einige Anforderungen der Authentifikatorzertifizierung erfüllt, wovon auch einige der verwendeten Lösungen den für Level 3+ notwendigen Kriterien entsprachen. Dies ist sowohl auf die verwendete Ausführungsumgebung, als auch auf die implementierte Software zurückzuführen. Jedoch wurden nicht für alle Anforderungen die zur Einstufung des Levels notwendigen Kriterien angegeben, wodurch eine Bewertung der gefundenen Lösungen nicht möglich war. Zudem wurden bei der Analyse der Umsetzung der Anforderungen sowohl Probleme auf Seiten der Implementierung als auch auf Seiten der Fingerabdruckkarte gefunden, weshalb eine Einschätzung des erreichten Authentifikatorlevels derzeit nicht möglich ist. Außerdem konnten einige Anforderungen nicht bewertet werden, da die dafür notwendigen Daten noch nicht vorliegen. Voraussetzung für eine erfolgreiche Authentifikatorzertifizierung nach Level 3(+) sind zudem die funktionale Zertifizierung des Tokens sowie die biometrische Zertifizierung des Sensors, welche nicht erfüllt werden konnten. Um die funktionale Zertifizierung zu erreichen, hätten alle Funktionalitäten des Authentifikators mit allen Testdiensten fehlerfrei funktionieren müssen. Außerdem konnte in dieser Arbeit nicht der von der FIDO Alliance bereitgestellte Selbsttest durchgeführt werden, da dieser nicht verfügbar war. Um die biometrische Zertifizierung des Fingerabdrucksensors durchführen lassen zu können, müsste zudem eine empirische Analyse der FAR sowie eine Analyse möglicher Angriffe auf den Sensor erfolgen (siehe Kapitel 7).

Bei dem im Rahmen dieser Arbeit entwickelten FIDO2-Applet ist dabei Verbesserungspotential in der Speicherung des dienstspezifischen privaten Schlüssels auf Seiten des Servers gegeben. Hier könnte in einer überarbeiteten Version der Anwendung beispielsweise ein Verfahren zur deterministischen Schlüsselableitung verwendet wer-

den, sodass nur das für die Ableitung benötigte kryptographische Material innerhalb der *credentialID* nach außen gegeben wird. Darüber hinaus sollte auch das weitere Schlüsselmanagement des Tokens betrachtet und ggf. überarbeitet werden. Auch die Konfigurationsmöglichkeit des Tokens durch den Benutzer sowie die Möglichkeit für Firmwareupdates sollte gegeben werden, um ein höheres Zertifizierungslevel erreichen zu können. Zusätzlich müssen für die Zertifizierung durch die FIDO Alliance die Metadaten des Tokens erfasst werden. Zudem stellt die verwendete Implementierung der Fingerabdrucklösung ein Problem für die Zertifizierung dar. Hier sollte in einer überarbeiteten Version der Fingerabdruckkarte die Kommunikation zwischen Fingerabdrucksensor und dem verbauten SE geschützt werden, um die entsprechenden Anforderungen für ein höheres Sicherheitslevel der Authentifikatorzertifizierung zu erfüllen (siehe Kapitel 7).

Literatur

[1] Salah Machani, Rob Philpott, Sampath Srinivas, John Kemp, and Jeff Hodges. FIDO UAF Architectural Overview. *FIDO Alliance Proposed Standard 02 February 2017*, 2017.

[2] Dirk Balfanz, Jakob Ehrensvard, and Juan Lang. FIDO U2F Raw Message Formats. *FIDO Alliance Proposed Standard 11 April 2017*, 2017.

[3] FIDO Alliance. FIDO® Certified. `https://fidoalliance.org/certification/fido-certified-products/` (aufgerufen am 24.10.2019), o. J..

[4] Dirk Balfanz, Alexei Czeskis, Jeff Hodges, J.C. Jones, Michael B. Jones, Akshay Kumar, Angelo Liao, Rolf Lindemann, and Emil Lundberg. Web Authentication: An API for accessing Public Key Credentials Level 1. `https://www.w3.org/TR/2019/REC-webauthn-1-20190304/` (aufgerufen am 03.10.2019), 2010.

[5] Christiaan Brand, Alexei Czeskis, Ehrensvärd, Michael B. Jones, Akshay Kumar, Rolf Lindemann, Adam Powers, and Johan Verrept. Client to Authenticator Protocol (CTAP). *Proposed Standard, January 30, 2019*, 2019.

[6] FIDO Alliance. FIDO Certification Program Policy. *Authenticator Certification Version 1.1.1 - May 2018*, 2018.

[7] Zhiqun Chen. *Java Card Technology for Smart Cards: Architecture and Programmer's Guide*. Addison-Wesley Professional, 2000.

[8] FIDO Alliance. FIDO2: Web Authentication (WebAuthn). `https://fidoalliance.org/fido2/fido2-web-authentication-webauthn/` (aufgerufen am 24.10.2019), o. J..

[9] Carsten Bormann and Paul E. Hoffman. Concise Binary Object Representation (CBOR). *RFC*, 7049:1–54, 2013.

[10] BSI. Passwörter. `https://www.bsi-fuer-buerger.de/BSIFB/DE/Empfehlungen/Passwoerter/passwoerter_node.html` (aufgerufen am 25.10.2019), o.J..

[11] Dinei Florencio and Cormac Herley. A Large-Scale Study of Web Password Habits. In *Proceedings of the 16th international conference on World Wide Web*, pages 657–666. ACM, 2007.

[12] Netcraft. September 2019 Web Server Survey. `https://news.netcraft.com/archives/2019/` (aufgerufen am 02.10.2019), 2019.

[13] Shikun Zhang, Sarah Pearman, Lujo Bauer, and Nicolas Christin. Why people (don't) use password managers effectively. In *SOUPS @ USENIX Security Symposium*. USENIX Association, 2019.

[14] Nora Alkaldi and Karen Renaud. Why do people adopt, or reject, smartphone password managers? 2016.

[15] Andreas Pashalidis and Chris J. Mitchell. A Taxonomy of Single Sign-On Systems. In *ACISP*, volume 2727 of *Lecture Notes in Computer Science*, pages 249–264. Springer, 2003.

[16] Europäische Union. Richtlinie (EU) 2015/2366 des Europäischen Parlaments und des Rates vom 25. November 2015 über Zahlungsdienste im Binnenmarkt, zur Änderung der Richtlinien 2002/65/EG, 2009/110/EG und 2013/36/EU und der Verordnung (EU) Nr. 1093/2010 sowie zur Aufhebung der Richtlinie 2007/64/EG, Amtsblatt der Europäischen Union vom 23.12. 20015. *Nr. L*, 337(35), 2015.

[17] Sampath Srinivas, Dirk Balfanz, Eric Tiffany, and Alexei Czeskis. Universal 2nd Factor (U2F) Overview. *FIDO Alliance Proposed Standard 11 April 2017*, 2017.

[18] FIDO Alliance. History of FIDO Alliance. `https://fidoalliance.org/overview/history/` (aufgerufen am 24.10.2019), o. J..

[19] FIDO Alliance. Functional Certification Program Policy. *Version 1.3.7 May 2019*, 2019.

[20] Meagan Karlsson. FIDO Biometric Component Certification Policy. *Final Document, April 02, 2019*, 2019.

[21] Viktor Taneski, Marjan Heričko, and Boštjan Brumen. Systematic Overview of Password Security Problems. *Acta Polytechnica Hungarica*, 16(3), 2019.

[22] Ameya Hanamsagar, S Woo, Christopher Kanich, and Jelena Mirkovic. How Users Choose and Reuse Passwords. *Information Sciences Institute*, 2016.

[23] Christian Reuter, Marc-André Kaufhold, and Jonas Klös. Benutzbare Sicherheit: Usability, Safety und Security bei Passwörtern. In *Mensch & Computer Workshopband*. Gesellschaft für Informatik e.V., 2017.

[24] Aaron L.-F. Han, Derek F. Wong, and Lidia S. Chao. Password Cracking and Countermeasures in Computer Security: A Survey. *arXiv preprint arXiv:1411.7803*, 2014.

[25] FIDO Alliance. FIDO Members. `https://fidoalliance.org/members/` (aufgerufen am 24.10.2019), o. J..

[26] FIDO Alliance. Liaison Partners. `https://fidoalliance.org/members/liaison/` (aufgerufen am 24.10.2019), o. J..

[27] Claudia Eckert. *IT-Sicherheit - Konzepte, Verfahren, Protokolle (10. Aufl.)*. Oldenbourg, 2018.

[28] Aleksandr Ometov, Sergey Bezzateev, Niko Mäkitalo, Sergey Andreev, Tommi Mikkonen, and Yevgeni Koucheryavy. Multi-Factor Authentication: A Survey. *Cryptography*, 2(1):1, 2018.

[29] Reza Fathi, Mohsen Amini Salehi, and Ernst L. Leiss. User-Friendly and Secure Architecture (UFSA) for Authentication of Cloud Services. In *CLOUD*, pages 516–523. IEEE Computer Society, 2015.

[30] Dipankar Dasgupta, Arunava Roy, and Abhijit Nag. *Advances in User Authentication*. Springer, 2017.

[31] Michele Orru and Giuseppe Trotta. Muraena - The Unexpected Phish [PowerPoint Präsentation]. `https://conference.hitb.org/hitbsecconf2019ams/materials/D2T1%20-%20Muraena%20-%20The%20Unexpected%20Phish%20-%20Michele%20Orru%20&%20Giuseppe%20Trotta.pdf` (aufgerufen am 13.10.2019), 2019.

[32] J. K. Mohsin, Liangxiu Han, Mohammad Hammoudeh, and Robert Hegarty. Two Factor Vs Multi-factor, an Authentication Battle in Mobile Cloud Computing Environments. In *ICFNDS*, pages 39:1–39:10. ACM, 2017.

[33] Nancie Gunson, Diarmid Marshall, Hazel Morton, and Mervyn A. Jack. User perceptions of security and usability of single-factor and two-factor authentication in automated telephone banking. *Computers & Security*, 30(4):208–220, 2011.

[34] Ding Wang, Debiao He, Ping Wang, and Chao-Hsien Chu. Anonymous two-factor authentication in distributed systems: Certain goals are beyond attainment. *IEEE Transactions on Dependable and Secure Computing*, 12(4):428–442, 2014.

[35] Aviel D. Rubin. Independent One-Time Passwords. *Computing Systems*, 9(1): 15–27, 1996.

[36] Neil Haller. The S/KEY One-Time Password System. *RFC*, 1760:1–12, 1995.

[37] Neil Haller, Craig Metz, Philip J. Nesser II, and Mike Straw. A One-Time Password System. *RFC*, 2289:1–25, 1998.

[38] David M'Raïhi, Mihir Bellare, Frank Hoornaert, David Naccache, and Ohad Ranen. HOTP: An HMAC-Based One-Time Password Algorithm. *RFC*, 4226:1–37, 2005.

[39] David M'Raïhi, Salah Machani, Mingliang Pei, and Johan Rydell. TOTP: Time-Based One-Time Password Algorithm. *RFC*, 6238:1–16, 2011.

[40] He Sun, Kun Sun, Yuewu Wang, and Jiwu Jing. TrustOTP: Transforming Smartphones into Secure One-Time Password Tokens. In *ACM Conference on Computer and Communications Security*, pages 976–988. ACM, 2015.

[41] David Silver, Suman Jana, Dan Boneh, Eric Yawei Chen, and Collin Jackson. Password Managers: Attacks and Defenses. In *USENIX Security Symposium*, pages 449–464. USENIX Association, 2014.

[42] Joshua Gray, Virginia N. L. Franqueira, and Yijun Yu. Forensically-Sound Analysis of Security Risks of Using Local Password Managers. In *RE Workshops*, pages 114–121. IEEE Computer Society, 2016.

[43] Paolo Gasti and Kasper Bonne Rasmussen. On the Security of Password Manager Database Formats. In *ESORICS*, volume 7459 of *Lecture Notes in Computer Science*, pages 770–787. Springer, 2012.

[44] Rui Wang, Shuo Chen, and XiaoFeng Wang. Signing Me onto Your Accounts through Facebook and Google: A Traffic-Guided Security Study of Commercially Deployed Single-Sign-On Web Services. In *IEEE Symposium on Security and Privacy*, pages 365–379. IEEE Computer Society, 2012.

[45] Dick Hardt. The OAuth 2.0 Authorization Framework. *RFC*, 6749:1–76, 2012.

[46] Nat Sakimura, John Bradley, Mike Jones, Breno de Medeiros, and Chuck Mortimore. OpenID Connect Core 1.0 incorporating errata set 1. *The OpenID Foundation, specification*, 2014.

[47] Brian Campbell, Chuck Mortimore, and Michael B. Jones. Security Assertion Markup Language (SAML) 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants. *RFC*, 7522:1–15, 2015.

[48] Clifford Neuman, Tom Yu, Sam Hartman, and Kenneth Raeburn. The Kerberos Network Authentication Service (V5). *RFC*, 4120:1–138, 2005.

[49] San-Tsai Sun and Konstantin Beznosov. The devil is in the (implementation) details: an empirical analysis of OAuth SSO systems. In *ACM Conference on Computer and Communications Security*, pages 378–390. ACM, 2012.

[50] Davit Baghdasaryan, Brad Hill, Roni Sasson, Jeff Hodges, and Ka Yang. FIDO UAF Authenticator-Specific Module API. *FIDO Alliance Proposed Standard 02 February 2017*, 2017.

[51] Rolf Lindemann. FIDO Metadata Service. *FIDO Alliance Proposed Standard 11 April 2017*, 2017.

[52] AuthenTrend. Fingerprint enabled Smart Badge. `https://authentrend.com/atkey-card/` (aufgerufen am 24.10.2019), o. J..

[53] Davit Baghdasaryan, Dirk Balafanz, Brad Hill, Jeff Hodges, and Ka Yang. FIDO UAF Protocol Specification. *FIDO Alliance Proposed Standard 02 February 2017*, 2017.

[54] Donghoon Chang, Sweta Mishra, Somitra Kumar Sanadhya, and Ajit Pratap Singh. On Making U2F Protocol Leakage-Resilient via Re-keying. *IACR Cryptology ePrint Archive*, 2017:721, 2017.

[55] Olivier Pereira, Florentin Rochet, and Cyrille Wiedling. Formal Analysis of the FIDO 1.x Protocol. In *FPS*, volume 10723 of *Lecture Notes in Computer Science*, pages 68–82. Springer, 2017.

[56] Christoforos Panos, Stefanos Malliaros, Christoforos Ntantogian, Angeliki Panou, and Christos Xenakis. A Security Evaluation of FIDO's UAF Protocol in Mobile and Embedded Devices. In *TIWDC*, volume 766 of *Communications in Computer and Information Science*, pages 127–142. Springer, 2017.

[57] Davit Baghdasaryan and Brad Hill. FIDO Security Reference. *FIDO Alliance Proposed Standard 11 April 2017*, 2017.

[58] Sanchari Das, Andrew Dingman, and L. Jean Camp. Why Johnny Doesn't Use Two Factor A Two-Phase Usability Study of the FIDO U2F Security Key. In *Financial Cryptography*, volume 10957 of *Lecture Notes in Computer Science*, pages 160–179. Springer, 2018.

[59] AuthenTrend. ATKEY.Card User Guide rev: 1.1. `https://authentrend.com/download/ATKey.card_Quick_Guide.pdf` (aufgerufen am 24.10.2019), o.J..

[60] Jim Schaad, Göran Selander, Derek Atkins, and Sean Turner. COSE Algorithms. `https://www.iana.org/assignments/cose/cose.xhtml#algorithms` (aufgerufen am 12.11.2019), 2019.

[61] Michael B. Jones. COSE and JOSE Registrations for WebAuthn Algorithms. `https://tools.ietf.org/html/draft-ietf-cose-webauthn-algorithms-02` (aufgerufen am 31.10.2019), 2019.

[62] Jan Camenisch, Manu Drijvers, Alec Edgington, Anja Lehmann, and Rainer Urian. FIDO ECDAA Algorithm. *FIDO Alliance Proposed Standard 02 February 2017*, 2017.

[63] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct Anonymous Attestation. *IACR Cryptology ePrint Archive*, 2004:205, 2004.

[64] FIDO Alliance. FIDO Privacy Principles. *Whitepaper vFeb2014*, 2014.

[65] Tim Bray. The JavaScript Object Notation (JSON) Data Interchange Format. *RFC*, 7159:1–16, 2014.

[66] Josef Langer and Michael Roland. *Anwendungen und Technik von Near Field Communication (NFC)*. Springer-Verlag, 2010.

[67] Stephanie Schuckers, Greg Cannon, Elhan Tabassi, Meagan Karlsson, and Elaine Netwon. FIDO Biometrics Requirements. *Final Document, June 06, 2019*, 2019.

[68] Dr. Joshua E. Hill and Douglas Biggs. FIDO Authenticator Security Requirements. *FIDO Alliance 05 September 2018*, 2018.

[69] Meagan Karlsson. FIDO Authenticator Metadata Requirements. *FIDO Alliance 29 June 2018*, 2018.

[70] Dr. Joshua E. Hill and Douglas Biggs. FIDO Authenticator Allowed Cryptography List. *FIDO Alliance 29 June 2018*, 2018.

[71] Laurence Lundblade and Meagan Karlsson. FIDO Authenticator Allowed Restricted Operating Environments List. *FIDO Alliance 29 June 2018*, 2018.

[72] Rolf Lindemann. FIDO Authenticator Security Requirements. *FIDO Alliance 05 September 2018*, 2018.

[73] FIDO Alliance. Certified Authenticator Levels. `https://fidoalliance.org/certification/authenticator-certification-levels/` (aufgerufen am 11.11.2019), o. J..

[74] NXP. JCOP 3 P60. Security Target Lite. `https://www.commoncriteriaportal.org/files/epfiles/[ST-Lite]%20ST-Lite_JCOP3_P60_v3.8.pdf` (aufgerufen am 12.11.2019), 2018.

[75] NXP. SmartMX2 P60: Single/Dual Contact Interface Controllers. `https://www.nxp.com/products/security-and-authentication/security-controllers/smartmx2-p60-single-dual-contact-interface-controllers:SMARTMX2-P60` (aufgerufen am 11.11.2019), o. J..

[76] STMicroelectronics. STM32L476xx. `https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&ved=2ahUKEwiG47OZpeXlAhUB4aQKHU1EANEQFjABegQIAxAC&url=https%3A%2F%2Fwww.st.com%2Fresource%2Fen%2Fdatasheet%2Fstm32l476je.pdf&usg=AOvVaw3iq8-Kh4SHhEaXi7TYni6m` (aufgerufen am 12.11.2019), 2019.

[77] Fingerprint Cards AB. Product Specification FPC1020. `http://www.shenzhen2u.com/doc/Module/Fingerprint/710-FPC1020_PB3_Product-Specification.pdf` (aufgerufen am 12.11.2019), 2014.

[78] Fingerprint Cards AB. FPC1020 Touch Fingerprint Sensor. `https://biometrics.mainguet.org/types/fingerprint/product/FPC/FPC_1020_flyer.pdf` (aufgerufen am 12.11.2019), o. J..

[79] Oracle Corporations. Java Card Protection Profile – Open Configuration. `https://www.oracle.com/technetwork/java/embedded/javacard/documentation/pp-jcs-open-392788.pdf` (aufgerufen am 12.11.2019), 2012.

[80] Jim Schaad, Göran Selander, Derek Atkins, and Sean Turner. COSE Elliptic Curves. https://www.iana.org/assignments/cose/cose.xhtml#elliptic-curves (aufgerufen am 12.11.2019), 2019.

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird.

Berlin, den 20. November 2019 .....