Institut für Informationssicherheit

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Über die (Un-) Sicherheit des W3C-WebAuthentication-Entwurfs: Eine Beschreibung und Sicherheitsanalyse

Miles Stötzner

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Ralf Küsters
Betreuer/in:	Dipl.-Inf. Guido Schmitz
Beginn am:	28. Juni 2018
Beendet am:	14. November 2018

Kurzfassung

Viele Dienste im Internet authentifizieren ihre Nutzer aufgrund der Benutzerfreundlichkeit durch ein Passwort. Dieses Passwort kann leicht von einem Angreifer in Erfahrung gebracht werden. Eine Lösung für dieses Problem ist der Einsatz mehrerer Faktoren.

WebAuthentication (WebAuthn) ist ein Entwurf eines Standards, der auf eine benutzerfreundliche Weise einem Nutzer ermöglicht, sich bei einer Relying Party mithilfe eines asymmetrischen Schlüsselpaares durch das Signieren einer von der Relying Party generierten Nonce, die sogenannte Challenge, unter Einsatz von mehreren Faktoren passwortlos zu registrieren und authentifizieren.

Während einer Registrierung oder Authentifizierung ist jeweils eine Relying Party, ein WebAuthn Client und ein Authenticator beteiligt. Der Authenticator ist für das Generieren des asymmetrischen Schlüsselpaares und für das Bereitstellen von Signaturen zuständig. Dabei muss jede Aktion von dem Nutzer autorisiert werden. Der WebAuthn Client befindet sich zwischen der Relying Party und dem Authenticator und ist für die Authentifizierung der Relying Party zuständig, sodass keine Aktion im Namen einer anderen Relying Party durchgeführt werden kann. Hierfür wird ein sicherer Kontext benötigt - d.h. die Verbindung ist über HTTPS gesichert. Während einer Registrierung wird ein asymmetrisches Schlüsselpaar generiert, dessen öffentlicher Schlüssel bei der Relying Party hinterlegt wird. Indem eine Signatur über die Challenge mit dem zugehörigen privaten Schlüssel generiert wird, kann sich ein Nutzer daraufhin authentifizieren.

Teil dieser Arbeit ist eine Implementierung einer Relying Party, die WebAuthn für die Authentifizierung ihrer Nutzer verwendet. Hierfür betrachten wir neben der Implementierung eines Severs die Verwendung der WebAuthentication API.

In einer informellen Sicherheitsanalyse untersuchen wir die Sicherheit des Protokolls, indem wir aufführen, wieso die während einer Registrierung bzw. Authentifizierung durchzuführenden Überprüfungen der einzelnen Parteien notwendig sind und welche Auswirkungen das Verwenden verschiedener Optionen von WebAuthn hat. Des Weiteren untersuchen wir die Kommunikationskanäle im Hinblick auf einen Man-in-the-Middle (MitM) und welche Gefahr von diesem ausgeht.

Während der Sicherheitsanalyse betrachten wir einen Angriff auf das Authentifizierungsverfahren und widerlegen die in der Spezifikation aufgeführte Behauptung, dass WebAuthn während einer Registrierung resistent gegenüber einem MitM sei, der den sicheren Kontext ignorieren kann. Beide Probleme wurden von den Autoren der Spezifikation anerkannt. Für den Angriff auf das Authentifizierungsverfahren wurde bereits eine Lösung gefunden.

Wir diskutieren, wieso das Authentifizierungsverfahren (nach Beheben unseres Angriffes) unter bestimmten Annahmen sicher ist und verweisen auf Probleme bezüglich der Privatsphäre und der für eine Registrierung oder Authentifizierung notwendige Zustimmung eines Nutzers. Zu den Annahmen zählen, dass der WebAuthn Client und Authenticator des Nutzers sowie die Relying Party sich konform verhalten, dass die Verbindung zwischen WebAuthn Client und Authentictor sicher ist und dass während der Registrierung kein erfolgreicher Angriff stattgefunden hat.

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Listings

Terminologie

AAGUID Die AAGUID identifiziert den Typ eines Authenticators und somit dessen Eigenschaften und Vertrauenswürdigkeit.

Anwesenheit einer Person Ein Authenticator kann vor Ausführen einer Operation die Anwesenheit einer Person durch eine einfache Interaktion testen - beispielsweise durch das Betätigen eines Knopfes.

Ascensus Ascensus ist die Relying Party der Demo-Implementierung.

Assertion Signature Die Assertion Signature wird während einer Authentifizierung generiert und signiert Informationen über den Authenticator und den WebAuthn Client. In diesen Informationen sind unter anderem die Challenge, die RPID und der Origin der aufrufenden Relying Party enthalten.

Attestation Statement Das Attestation Statement ist Teil eines Attestation Objects und enthält die Attestation Signature sowie Informationen zur Verifizierung.

Attestation Signature Die Attestation Signature wird während einer Registrierung generiert und signiert Informationen über den Authenticator und den WebAuthn Client. In diesen Informationen sind unter anderem der neu generierte öffentliche Schlüssel, die Challenge, die RPID und der Origin der aufrufenden Relying Party enthalten.

Attestation Key Der Attestation Key ist der Schlüssel, der zum Erzeugen einer Attestation Signature genutzt wird.

Attestation Object Das Attestation Object wird während einer Registrierung erzeugt. Es beinhaltet neben dem öffentlichen Schlüssel des Nutzers auch eine Attestation Statement und Informationen zur Verifizierung.

Authenticator Ein Authenticator ist zuständig für das Verwalten von asymmetrischen Schlüsselpaaren und das Bereitstellen von Signaturen.

Authenticator Data Die Authenticator Data enthalten unter anderem eine RPID und Markierungen bezüglich der Anwesenheit einer Person bzw. der Verifizierung des Nutzers. Während einer Registrierung sind zusätzlich eine AAGUID und ein öffentlicher Schlüssel enthalten.

Authentifizierung Eine Authentifizierung ist der Vorgang, bei dem ein zuvor registrierter Nutzer seine Identität gegenüber einer Relying Party mithilfe eines zuvor registrierten öffentlichen Schlüssels in Zusammenarbeit mit einem WebAuthn Client und einem Authenticator beweist.

BLE Bluetooth Low Energy (BLE) ist eine Verbindungsart, die zwischen einem WebAuthn Client und Authenticator eingesetzt werden kann.

Challenge Die Challenge ist eine von einer Relying Party generierte Nonce, die während einer Registrierung bzw. Authentifizierung Teil einer Attestation Signature bzw. Assertion Signature ist.

Client Data Die Client Data enthalten die Challenge, den Origin und den Status des Token Binding einer Registrierung bzw. Authentifizierung und sind Teil der Attestation Signature bzw. Assertion Signature.

Credential ID Eine Credential ID ist eine von einem Authenticator generierte Nonce, die ein asymmetrisches Schlüsselpaar eindeutig identifiziert.

Erweiterung Während einer Registrierung bzw. einer Authentifizierung kann eine Relying Party zusätzlich Operationen durchführen lassen. Beispielsweise kann die Relying Party die geographische Position des Authenticator anfragen.

MitM Ein Man-in-the-Middle (MitM) ist ein Angreifer, der sich zwischen zwei Parteien befindet und dabei Nachrichten lesen und ändern kann.

Multi-Faktor-Authentifizierung Bei einer Multi-Faktor-Authentifizierung beweist ein Nutzer über mindestens zwei der folgenden Faktoren seine Identität: etwas, dass der Nutzer besitzt, weiß oder ist. In unserem Fall besitzt der Nutzer einen Authenticator, auf dem er sich beispielsweise über etwas, das er weiß oder ist, verifiziert.

NFC Near Field Communication (NFC) ist eine Verbindungsart, die zwischen einem WebAuthn Client und Authenticator eingesetzt werden kann.

Nonce Eine Nonce ist eine zufällig generierte Zahl.

Origin Ein Origin ist ein Tupel bestehend aus Schema, Host, Port und Domain - beispielsweise (“https://”, “example.ascensus.com”, 1337, null) bzw. “https://example.ascensus.com:1337” [WHA18].

Registrierung Eine Registrierung ist der Vorgang, bei dem ein Nutzer einen öffentlichen Schlüssel bei einer Relying Party in Zusammenarbeit mit einem WebAuthn Client und einem Authenticator hinterlegt.

Relying Party Eine Relying Party nutzt WebAuthn, um Nutzer zu registrieren und zu authentifizieren. Sie wird über eine RPID identifiziert und kann mehrere Origins besitzen.

RPID Eine Relying Party Identifier (RPID) identifiziert eine Relying Party. Sie wird während der Registrierung und Authentifizierung genutzt, um sicherzustellen, dass eine Relying Party nur Zugriff auf die von ihr registrierten Nutzer hat. Die Relying Party Identifier (RPID) leitet sich aus den Origins der Relying Party ab (siehe Schritt 2.3.3 der Registrierung). Dadurch kann die Relying Party mehrere Origins mit der gleichen RPID nutzen.

Signature Counter Ein Signature Counter ist aein Zähler, der immer erhöht wird, wenn eine Assertion Signature erzeugt wird.

Token Binding Token Binding ist ein Protokoll, um eine TLS Verbindung eindeutig zu identifizieren [PNB18]. Dadurch kann ein Cookie an eine bestimmte TLS Verbindung gebunden werden.

TPM Ein Trusted Platform Module (TPM) ist eine abgeschottete Komponente eines System, die für kryptographischen Operationen eingesetzt wird.

U2F FIDO U2F ist ein Second-Faktor-Protokoll [SB17]. Ein Nutzer authentifiziert sich beispielsweise über die Eingabe eines Passwortes und zusätzlich als zweiten Faktor über das Bestätigen der Authentifizierung auf seinem Authenticator.

USB Universal Serial Bus (USB) ist eine Verbindungsart, die zwischen einem WebAuthn Client und Authenticator eingesetzt werden kann.

User Handle Ein User Handle ist eine von der Relying Party generiert Nonce, die einen Nutzer eindeutig identifiziert.

Verifizierung des Nutzers Ein Authenticator kann den Nutzer vor Ausführen einer Operation beispielsweise über einen Fingerabdruckscan oder die Eingabe eines PIN verifizieren.

W3C Das World Wide Web Konsortium (W3C) ist eine internationale Vereinigung mit dem Ziel Protokolle für die Verwendung im Internet zu standardisieren.

WebAuthentication API Die WebAuthentication API wird vom WebAuthn Client verkörpert und ist die Schnittstelle zwischen Relying Party und Authenticator.

WebAuthn WebAuthentication (WebAuthn) ist ein Entwurf eines Standards, das auf eine benutzerfreundliche Weise einem Nutzer ermöglicht, sich bei einer Relying Party mithilfe eines asymmetrischen Schlüsselpaares durch das Signieren einer von der Relying Party generierten Nonce, die sogenannte Challenge, unter Einsatz von mehreren Faktoren passwortlos zu registrieren und authentifizieren.

WebAuthn Client Ein WebAuthn Client implementiert die WebAuthentication API. Er leitet Nachrichten weiter und fungiert somit als Brücke zwischen Relying Party und Authenticator.

1 Einleitung

Alice möchte ihre Finanzen online verwalten und registriert sich daher auf der Internetseite ihrer Bank. Da Alice auf vielen verschiedenen Internetseite registriert ist und sich nicht für jede einzelne Seite ein Passwort merken kann, nutzt sie überall den gleichen Nutzernamen und das gleiche Passwort. Mit diesen Zugangsdaten hat sich Alice auch auf Eves Internetseite registriert. Eve hat unehrliche Absichten und versucht sich mit den Zugangsdaten bei Alice Bank einzuloggen, die Alice während ihrer Registrierung bei Eve angegeben hat. Da die Zugangsdaten korrekt sind, kann sich Eve erfolgreich bei der Bank als Alice ausgeben und somit Alice Finanzen einsehen.

Eve könnte auch Alice Zugangsdaten mithilfe eines Phishing Angriffes erlangen. Dabei verleitet sie Alice beispielsweise dazu, die Zugangsdaten zu ihren Finanzen fälschlicherweise auf Eves Internetseite einzugeben.

Das aufgeführte Beispiel zeigt folgende Problematiken:

- die Merkbarkeit und die Wiederverwendbarkeit eines Passwortes,
- das Nutzen von nur einem Faktor während der Registrierung und Authentifizierung
- die Anfälligkeit gegenüber Phishing Angriffen.

Ein Nutzer kann sich nur schwer viele Passwörter einprägen. Daher tendieren viele Nutzer dazu ein Standardpasswort zu haben, welches sie bei verschiedenen Internetseiten nutzen. Eine unehrliche Internetseite kann diese Zugangsdaten daher bei anderen Internetseiten nutzen, um sich als der Nutzer auszugeben - insbesondere wenn das Passwort der einzige eingesetzte Faktor ist. Eine Lösung für dieses Problem ist die Verwendung mehrerer Faktoren:

- etwas, das du weißt (Passwort, PIN ...)
- etwas, das du besitzt (Karte, Schlüssel ...)
- etwas, das du bist (Fingerabdruckscan, Iris-Scan ...)

WebAuthentication (WebAuthn) ist ein vom World Wide Web Konsortium (W3C) spezifizierter Entwurf eines Standards, der auf eine benutzerfreundliche Weise einem Nutzer ermöglicht, sich bei einer Relying Party mithilfe eines asymmetrischen Schlüsselpaares durch das Signieren einer von der Relying Party generierten Nonce unter Einsatz von mehreren Faktoren passwortlos zu registrieren und authentifizieren. Dabei muss darauf geachtet werden, dass die Benutzerfreundlichkeit unter einem solchen Protokoll nicht leidet. Ist beispielsweise das Verwenden des Authentifizierungsverfahren für einen durchschnittlichen Nutzer ohne technisches Hintergrundwissen zu kompliziert, wird das Verfahren sich trotz erhöhter Sicherheit nicht durchsetzen können.

In dieser Arbeit befassen wir uns mit der Beschreibung, Implementierung und einer informellen Sicherheitsanalyse von WebAuthn.

In Kapitel 2 werden wir uns zunächst eine Überblick über WebAuthn verschaffen. Dabei gehen wir nicht auf technische Details ein, sondern werden das Protokoll auf einer abstrahierten konzeptuellen Ebene beschreiben. Hierfür betrachten wir die beteiligten Parteien und die einzelnen Schritte während einer Registrierung bzw. Authentifizierung.

In Kapitel 3 betrachten wir eine Implementierung einer Relying Party als Node.js Server, die Nutzer unter Verwendung von WebAuthn und eines Yubikey erfolgreich registriert und authentifiziert. Dabei steht die Verwendung der WebAuthentication API und die von der Relying Party durchzuführenden Überprüfungen im Mittelpunkt.

In Kapitel 4 folgt eine informelle Sicherheitsanalyse. Hierfür legen wir zunächst verschiedene Anforderungen an das Protokoll in Form von Sicherheitszielen fest, treffen für die Analyse geltende Annahmen und beschreiben die Bedrohungslage. Wir untersuchen die Sicherheit, indem wir aufführen, wieso die während einer Registrierung bzw. Authentifizierung durchzuführenden Überprüfungen der einzelnen Parteien notwendig sind und welche Auswirkungen das Verwenden verschiedener Optionen von WebAuthn auf die Sicherheitsziele hat. Des Weiteren untersuchen wir die Kommunikationskanäle im Hinblick auf einen Man-in-the-Middle und welche Gefahr von diesem ausgeht. Anschließend diskutieren wir aufgrund der Analyse aus, welche Sicherheitsziele gewährleistet werden können.

Während der Sicherheitsanalyse betrachten wir in Kapitel 4.2.1.10 einen Angriff auf das Authentifizierungsverfahren und widerlegen in Kapitel 4.2.2.2 die in der Spezifikation aufgeführte Behauptung, dass WebAuthentication während einer Registrierung resistent gegenüber einem Man-in-the-Middle sei, der den sicheren Kontext ignorieren kann. Beide Probleme wurden von den Autoren der Spezifikation anerkannt. Für den Angriff auf das Authentifizierungsverfahren wurde bereits eine Lösung gefunden.

In Kapitel 5 betrachten wir rückblickend die Ergebnisse der vorhergegangenen Kapitel und führen uns vor Augen, welche Probleme durch WebAuthn gelöst werden und welche nicht - insbesondere welche Probleme durch das Verwenden des Protokolls hinzugekommen sind. Des Weiteren führen wir auf, welche Schritte als Nächstes unternommen werden sollten.

1.1 Verwandte Arbeiten

I.B. Guirat und H. Halpin [GH18] haben eine formale Sicherheitsanalyse von WebAuthn veröffentlicht. In dieser modellieren sie das Protokoll in ProVerif und zeigen anhand dessen, dass WebAuthn gegenüber einem Angreifer sicher ist, der sich zwischen der Relying Party und dem WebAuthn Client befindet und die Kontrolle über das Netzwerk hat. Dabei merken die Autoren allerdings an, dass ihr Modell das Protokoll sehr vereinfacht und ihre Analyse damit nur als Startpunkt dienen soll. Darüber hinaus weisen sie auf ein Problem mit der Privatsphäre des Nutzers hin: Der öffentliche Schlüssel, der während einer Registrierung zum Verifizieren eingesetzt wird, kann den Nutzer als Besitzer verschiedener Accounts bei der gleichen oder einer weiteren Relying Party identifizieren.

Paragon Initiative Enterprises [Par18] hat auf mehrere krypotgraphische Probleme bei WebAuthn hingewiesen. Beispielsweise bemängeln sie, dass RSA PKCS1v1.5 unterstützt wird und dass das unterstütze Signierverfahren ECDAA noch nicht ausreichend analysiert worden ist. Zu einem späteren Zeitpunkt hat Paragon Initiative Enterprises ein Kommentar diesbezüglich abgegeben und

betont, dass die aufgeführten Probleme leicht durch eine Aktualisierung der Spezifikation behoben werden können und diese somit keine Kritik an dem Konzept von WebAuthn sein sollen. In dieser Arbeit gehen wir allerdings nicht auf solche technischen Details ein.

In der Spezifikation der WebAuthentication API [BCH18] wird nicht nur das Konzept des Protokolls aufgeführt, sondern auch Bedenken hinsichtlich der Sicherheit und der Privatsphäre eines Nutzers. Diese Bedenken werden in dieser Arbeit ebenfalls diskutiert.

FIDO Alliance analysiert in *FIDO Security Reference* [Lin18] die Sicherheit von U2F. Die in dieser Referenz aufgeführten Punkte sind auch auf WebAuthn anwendbar, da WebAuthn abwärtskompatibel ist und daher Authenticatoren im Rahmen von WebAuthn eingesetzt werden können, die U2F unterstützen.

Jacomme und Kremer [JK18] beschreiben in ihrer Arbeit eine umfassende Bedrohungslage für eine Multi-Faktor-Authentifizierung und untersuchen daraufhin unter anderem die Sicherheit von U2F, indem sie das Protokoll mit der gegebenen Bedrohungslage in ProVerif modellieren. Zu der Bedrohungslage gehören neben einem Netzwerkangreifer auch die an einer Authentifizierung beteiligte Parteien, die möglicherweise infiziert sind. In ihrer Analyse kommen sie zu dem Schluss, dass U2F ein sicheres Verfahren ist, sofern der Computer und Security Key des Nutzers vertrauenswürdig und somit nicht infiziert sind.

2 Beschreibung

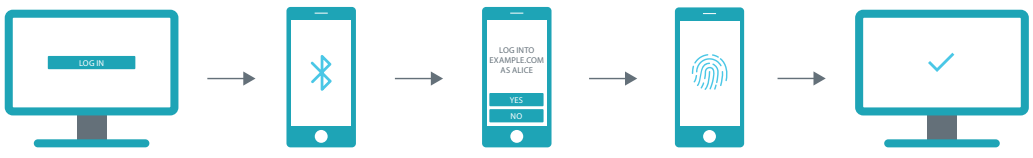
In diesem Kapitel befassen wir uns mit der Beschreibung von WebAuthn, die inhaltlich auf der Spezifikation [BCH18] vom 7. August 2018 des W3C basiert. Dabei soll diese keine detaillierte technische Beschreibung sein, sondern eine Abstraktion zum besseren Verständnis als Grundlage für die in Kapitel 4 folgende Analyse.

Zunächst schauen wir uns in Kapitel 2.1 verschiedene Anwendungsmöglichkeiten an, in denen WebAuthn genutzt werden kann. Es folgt eine Übersicht der Aufgaben und Eigenschaften der beteiligten Parteien: Relying Party, WebAuthn Client und Authenticator. Danach betrachten wir in Kapitel 2.3 den Prozess einer Registierung eines Nutzers und wie die beteiligten Parteien während dieser zusammenarbeiten. Wie ein registrierter Nutzer sich gegenüber einer Relying Party authentifizieren kann, behandeln wir in Kapitel 2.4. Abschließend untersuchen wir mögliche Erweiterungen und Fälle eines Schlüssel-Widerrufs.

2.1 Anwendungsmöglichkeiten

In diesem Abschnitt betrachten wir verschiedene Anwendungsmöglichkeiten, in denen WebAuthn genutzt wird, um einen Nutzer zunächst zu registrieren und anschließend zu authentifizieren. Diese sind aufgrund der Plattformunabhängigkeit der Spezifikation von WebAuthn sehr variabel und vielfältig. Dabei steht insbesondere eine passwortlose Authentifizierung im Mittelpunkt. In folgenden Beispielen betrachten wir die Anwedungsmöglichkeiten:

Registrierung Alice möchte sich bei einer Website registrieren, die sie über ihren Browser an ihrem Computer besucht. Die Website fordert Alice auf, den Account zusammen mit einem Nutzernamen und einem Smartphone anstelle eines Passwortes zu registrieren. Alice verbindet ihr Smartphone über Bluetooth mit ihrem Computer und bestätigt die Registrierungsanfrage auf ihrem Smartphone durch einen Fingerabdruckscan. Alice ist nun auf der Website registriert.



1. Anfrage an Website | 2. Bluetooth Verbindung | 3. Akzeptieren der Anfrage | 4. Authorizieren der Anfrage | 5. Erfolgreiche Authentifizierung

Abbildung 2.1: Beispiel einer Authentifizierung

Authentifizierung Um sich nun bei der Website aus dem vorherigen Szenario einzuloggen, besucht Alice das Login-Fenster der Website und verbindet ihr Smartphone über Bluethooth mit ihrem Computer. Daraufhin erscheint eine Anfrage auf ihrem Smartphone, die sowohl den Namen der Website als auch ihren Nutzernamen beinhaltet. Nach Bestätigen dieser Anfrage durch einen Fingerabdruckscan wird Alice erfolgreich an ihrem Computer eingeloggt (siehe Abbildung 2.1).

Neues Gerät registrieren Alice loggt sich das erste Mal mithilfe ihres Smartphones an ihrem Laptop bei der zuvor registrierten Website ein. Die Website bietet die Möglichkeit an, zusätzlich zu ihrem Smartphone den Laptop zu registrieren. Hierfür bestätigt Alice die Anfrage durch einen Fingerabdruckscan an ihrem Laptop und kann nun zukünftigt sich an diesem ohne ihr Smartphone authentifizieren.

In den aufgeführten Beispielen authentifiziert sich Alice durch eine **Multi-Faktor-Authentifizierung**, da sie zum einen im Besitz des Smartphones ist und zum anderen die Authentifizierungsanfrage mithilfe ihres Fingerabdrucks an ihrem Smartphone autorisiert.

Folgende Anwendungsbereiche sind erwähnenswert, um die vielfältigen Einsatzmöglichkeiten zu verdeutlichen. Dabei werden die ersten zwei nicht explizit in der Spezifikation erwähnt. WebAuthn kann des Weiteren genutzt werden, um

- Wähler mittels Personalausweis für Online-Wahlen zu authentifizieren. Dabei besitzt der Personalausweis, wie beispielsweise die *Mastercard Biometric Card* [Mas17], einen eingebauten Fingerabdruckscanner.
- verschiedene Autos mittels eines einzigen Autoschlüssels bedienen. Aktuell existiert beispielsweise die *Connected Vehicle API* [Mer18a]. Mit Hilfe dieser API haben Mercedes-Kunden über eine Webanwendung die volle Kontrolle über ihr Auto und können dieses unter anderem ver- und entriegeln, die Position bestimmen oder den Motor starten. Dabei wird für die Authentifizerung ein Nutzername und ein Passwort verwendet [Mer18b].
- mit einem (Studierenden-)Ausweis zu bezahlen. Dabei kann WebAuthn für die Authentifizerung und die *Payment Handler API* in Kombination mit der *Payment Request API* für die tatsächliche finanzielle Transaktion verwendet werden [W3C18].

2.2 Parteien

Während einer Registrierung oder Authentifizierung sind neben einem Nutzer folgende Parteien beteiligt: eine Relying Party, ein WebAuthn Client und ein Authenticator. In diesem Kapitel betrachten wir die Rolle einer jeden Partei und welche Aufgaben diese übernehmen.

Abbildung 2.2: WebAuthentication Übersicht

2.2.1 Relying Party

Eine Relying Party (siehe Abbildung 2.2) nutzt WebAuthn, um Nutzer zu registrieren und zu authentifizieren. Dabei besitzt jede Relying Party eine RPID, über die sie eindeutig identifiziert werden kann. Die RPID leitet sich aus den Origins der Relying Party ab (siehe Schritt 2.3.3 der Registrierung). Dadurch kann die Relying Party kann mehrere Origins mit der gleichen RPID nutzen.

Folgende vereinfacht dargestellte Operationen stehen einer Relying Party zur Verfügung:

Registrierung Um einen Nutzer zu registrieren, stellt die Relying Party eine Anfrage zusammen mit einer Challenge, RPID und weiteren Informationen an einen WebAuthn Client. Eine Challenge ist eine von der Relying Party generierte Nonce, die von dem Nutzer signiert werden soll. Bei erfolgreicher Ausführung erhält die Relying Party eine von einem Authenticator signierte Antwort. In dieser Antwort sind Informationen über den Authenticator und den WebAuthn Client enthalten - insbesondere die Challenge, die RPID und der Origin der Relying Party sowie ein öffentlicher Schlüssel. Der öffentliche Schlüssel wird im Zusammenhang mit dem Nutzer für zukünftige Authentifizierungen abgespeichert. Mithilfe der Signatur kann die Relying Party die Vertrauenswürdigkeit des öffentlichen Schlüssels prüfen.

Authentifizierung Um einen Nutzer zu authentifizieren, stellt die Relying Party eine Anfrage zusammen mit einer Challenge, RPID und weiteren Informationen an den WebAuthn Client. Bei erfolgreicher Ausführung erhält die Relying Party eine signierte Antwort. In der Antwort sind Informationen über den Authenticator und den WebAuthn Client enthalten - insbesondere die Challenge, die RPID und der Origin der Relying Party. Falls die Signatur mit dem im Zusammenhang mit dem Nutzer abgespeicherten öffentlichen Schlüssel verifiziert werden kann und die Informationen den zu erwartenden Werten entspricht, ist der Nutzer der Relying Party gegenüber authentifiziert.

Eine Relying Party wird genau dann als *konform* bezeichnet, wenn sie sich wie in der Spezifikation beschrieben verhält. Die Konformität schützt sie vor nicht-konformen, sich falsch verhaltenden und feindseligen Parteien.

Beispielsweise nutzt die Relying Party “Universität Stuttgart” mit dem Origin “https://uni-stuttgart.de” als RPID “uni-stuttgart.de”.

2.2.2 WebAuthn Client

Ein WebAuthn Client (siehe Abbildung 2.2) besteht aus einem Browser und dem zugehörigen Betriebssystem und implementiert die WebAuthentication API. Indem der WebAuthn Client als Brücke zwischen einer Relying Party und einem Authenticator dient, kann die Relying Party auf Veranlassung eines Nutzers Operationen auf dem Authenticator ausführen lassen. Dabei stellt der WebAuthn Client sicher, dass keine Aktionen im Namen einer anderen Relying Party durchgeführt werden können.

Die WebAuthentication API dient der Verwaltung von asymmetrischen Schlüsseln, um mit diesen Nutzer gegenüber Relying Parties zu authentifizieren. Dafür stehen einem WebAuthn Client unter anderem folgende vereinfacht dargestellte Operationen zur Verfügung - dabei verfolgen die Operationen während einer Registrierung bzw. Authentifizierung das gleiche Konzept:

create, get Der WebAuthn Client erhält von einer Relying Party Optionen, um einen Nutzer zu registrieren (*create*) bzw. zu authentifizieren (*get*). Diese Anfrage leitet der WebAuthn Client nach einer Verifizierung zusammen mit Informationen über den WebAuthn Client an alle verfügbaren Authenticatoren weiter. Die Informationen enthalten die Challenge, den Origin und die RPID der aufrufenden Relying Party. Bei einer erfolgreichen Antwort sendet der WebAuthn Client die erhaltene Antwort an die Relying Party zurück.

Der WebAuthn Client stellt aus Sicherheitsgründen die WebAuthentication API nur in einem sicheren Kontext, bei einer über HTTPS gesicherten Verbindung, zur Verfügung. Die Sicherheitsbedenken diesbezüglich werden in Kapitel 4 behandelt.

Ein WebAuthn Client wird genau dann als *konform* bezeichnet, wenn er sich wie in der Spezifikation beschrieben verhält. Die Konformität schützt ihn vor nicht-konformen, sich falsch verhaltenden und feindseligen Parteien.

Als WebAuthn Client kann beispielsweise Firefox 60 [PP18] oder Microsoft Edge [NMC17] zusammen mit dem Betriebssystem genutzt werden.

2.2.3 Authenticator

Ein Authenticator (siehe Abbildung 2.2) ist zuständig für das Verwalten von privaten Schlüsseln und das Generieren von Signaturen. Ein privater Schlüssel wird unter anderem zusammen mit einer RPID abgespeichert. Mithilfe dieser RPID stellt ein Authenticator sicher, dass der private Schlüssel nur für Operationen im Kontext der zugehörigen Relying Party genutzt wird.

Folgende vereinfacht dargestellte Operationen stehen einem Authenticator zur Verfügung:

authenticatorMakeCredential Der Authenticator generiert ein asymmetrisches Schlüsselpaar. Das Schlüsselpaar besteht aus einem privaten und einem öffentlichen Schlüssel. Der private Schlüssel wird im Zusammenhang mit einer RPID abgespeichert. Der öffentliche Schlüssel ist in den Informationen über den Authenticator enthalten, die zusammen mit den Informationen über den WebAuthn Client signiert zurückgegeben werden - die sogenannte **Attestation Signature**.

Sofern von der Relying Party verlangt, wird vor dem Ausführen dieser Operation eine Verifizierung des Nutzers durchgeführt. Ansonsten wird lediglich die Anwesenheit einer Person geprüft.

Alle nötigen Informationen werden von dem aufrufenden WebAuthn Client bzw. der aufrufenden Relying Party übergeben.

authenticatorGetAssertion Der Authenticator nutzt eine RPID, um den privaten Schlüssel eines registrierten Nutzers zu finden. Mit diesem werden die Informationen über den Authenticator und den WebAuthn Client signiert und zurückgegeben - die sogenannte **Assertion Signature**.

Sofern von der Relying Party verlangt, wird vor dem Ausführen dieser Operation eine Verifizierung des Nutzers durchgeführt. Ansonsten wird lediglich die Anwesenheit einer Person geprüft.

Alle nötigen Informationen werden von dem aufrufenden WebAuthn Client bzw. der aufrufenden Relying Party übergeben.

Jeder Authenticator besitzt eine vom Hersteller beliebig gewählte AAGUID, welche das Modell des Authenticators identifiziert. Das bedeutet, dass Authenticatoren des gleichen Modells die gleiche AAGUID besitzen. Sie kann von einer Relying Party genutzt werden, um gewisse Eigenschaften oder das Vertrauenslevel eines Authenticators zu ermitteln.

Ein Authenticator sollte einen der folgenden streng monoton wachsenden Zähler implementieren, den sogenannten Signature Counter. Der Wert des Zählers ist ebenfalls in den Signaturen während einer Registrierung oder Authentifizierung enthalten:

- Ein Signature Counter *pro Schlüsselpaar*, der bei jeder Authentifizierung im Zusammenhang mit dem jeweiligen Schlüsselpaar erhöht wird.
- Ein *globaler* Signature Counter, der bei jeder Authentifizierung erhöht wird.

Jeder Authenticator muss die Anwesenheit einer Person testen können. Ein Authenticator, der zusätzlich eine Verifizierung des Nutzers durchführt, stellt eine Multi-Faktor-Authentifizierung bereit: Der erste Faktor ist der Besitz des Authenticator, der zweite ist die Verifizierung des Nutzers.

Der Nachrichtenaustausch zwischen einem WebAuthn Client und Authenticator ist abhängig davon, ob dieser ein *platform* oder *cross-platform* Authenticator ist. Erstere sind fest in dem WebAuthn Client verbaut und nutzen eine plattformspezifische Kommunikation - beispielsweise als TPM. Letztere sind meistens bewegliche externe Geräte, die eine standardisierte Kommunikation nutzen - beispielsweise USB, NFC oder BLE. Die Spezifikation schreibt nicht vor, welche Art von Kommunikation verwendet werden muss.

Ein Authenticator wird genau dann als *konform* bezeichnet, wenn er sich wie in der Spezifikation beschrieben verhält. Die Konformität schützt ihn vor nicht-konformen, sich falsch verhaltenden und feindseligen Parteien.

Als Authenticator kann beispielsweise ein FIDO U2F Security Key, Smartphone oder Windows Hello [NMC17] genutzt werden.

2.3 Registrierung

In diesem Abschnitt wird der Prozess einer Registrierung näher beschrieben. Eine Registrierung ist der Vorgang, bei dem ein Nutzer einen Account bei einer Relying Party in Zusammenarbeit mit einem WebAuthn Client und einem Authenticator erstellt und dabei einen öffentlichen Schlüssel bei der Relying Party hinterlegt. Besitzt der Nutzer bereits einen Account, kann dieser eine zusätzliche Möglichkeit der Authentifizierung hinzufügen. Mit Hilfe der bei der Registrierung gespeicherten Informationen kann sich ein Nutzer im Nachhinein der Relying Party gegenüber authentifizieren (siehe Kapitel 2.4).

Eine Übersicht der einzelnen Schritte ist in Abbildung 2.3 dargestellt. Dabei bestehen die einzelnen Schritte aus den in den vorherigen Kapiteln vorgestellten Operationen.

1. **Registrierungsanfrage**

   Der Prozess der Registrierung wird eingeleitet, indem ein Nutzer eine Registierungsanfrage mithilfe seines WebAuthn Clients an eine Relying Party stellt. Teil der Anfrage kann ein Nutzername oder weitere Informationen sein, welche die Relying Party benötigt. Dieser Schritt wird in der Spezifikation nicht näher behandelt.

2. **Relying Party sendet Optionen**

   Die Relying Party erhält die Registrierungsanfrage und sendet folgende Optionen an den WebAuthn Client:

   rp beschreibt die Relying Party anhand eines Namens und einer RPID.

   user beschreibt den zu registrierenden Nutzer und beinhaltet einen Namen, einen Anzeigenamen und einen User Handle. Der Name soll dazu beitragen, verschiedene Accounts eines Nutzers einfacher für diesen differenzieren zu können - beispielsweise “alice@alison.com”. Der Anzeigename dient zum Anzeigen einer nutzerfreundlichen Oberfläche - beispielweise “Alice Alison”. Der User Handle hingegen ist eine neu generierte Nonce, die den Nutzer eindeutig identifiziert.

   challenge enthält eine neu generierte Challenge.

   pubKeyCredParams ist eine Liste mit Paaren von erlaubten Algorithmen zum Generieren eines asymmetrischen Schlüsselpaares - beispielsweise ES256.

   timeout gibt an, nach welcher Zeit die Operation abgebrochen werden soll.

   excludeCredentials enthält Credential IDs, bei denen ein Authenticator abbrechen muss, sofern eine dieser Credential IDs auf dem Authenticator registriert ist. Dadurch kann die Relying Party das mehrfache Nutzen eines Authenticators für verschiedene Schlüssel eines einzelnen Nutzers einschränken.

   Eine Credential ID ist eine von einem Authenticator generierte Nonce, die ein asymmetrisches Schlüsselpaar eindeutig identifiziert.

   authenticatorAttachment erlaubt entweder *platform* oder *cross-platform* Authenticatoren.

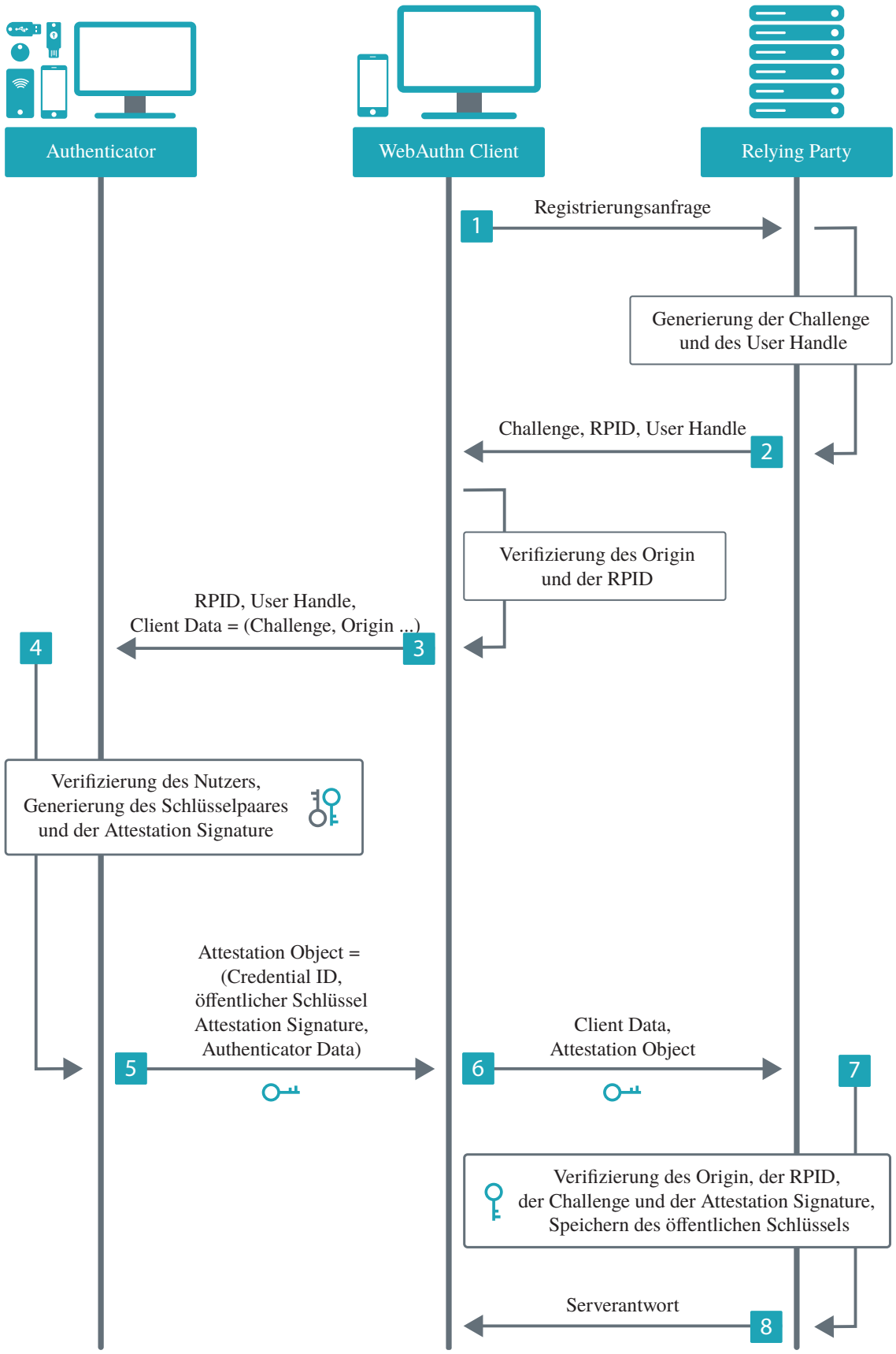


Abbildung 2.3: Einzelne Schritte während einer Registrierung

requireResidentKey gibt an, ob der Authenticator den zu erstellenden privaten Schlüssel zusammen mit weiteren Informationen speichern soll oder nicht.

userVerification bestimmt, ob eine Verifizierung des Nutzers auf dem Authenticator durchgeführt, bevorzugt oder verhindert werden muss.

attestation definiert Anforderungen an ein Attestation Statement. Dabei existieren drei mögliche Optionen: Bei *none* erwartet die Relying Party kein Attestation Statement, bei *direct* das vom Authenticator unveränderte und bei *indirect* ein verifizierbares Attestation Statement.

extensions ist eine Liste mit Erweiterungen, die zusätzlich während der Registrierung ausgeführt werden sollen.

Die Standardeinstellungen von WebAuthn besagen: Es wird kein Attestation Statement gefordert, eine Verifizierung des Nutzers wird bevorzugt, der private Schlüssel muss nicht auf dem Authenticator gespeichert werden und es werden keine Erweiterungen ausgeführt.

3. WebAuthn Client übergibt Optionen an Authenticator

Der WebAuthn Client erhält die oben beschriebenen Optionen und startet die *create* Operation, bei der zunächst eine Validierung und der Aufruf der Authenticatoren durchgeführt wird. Falls ein Verifizierungsschritt fehlschlägt, wird die Registrierung abgebrochen.

Zunächst wird der Origin der Relying Party und die erhaltene RPID geprüft: Der Origin der Relying Party wird durch den Origin des aufrufenden Skriptes bestimmt. Dieses muss den gleichen Origin wie all seine Eltern besitzen (siehe *sameOriginWithAncestors* [Wes18]). Des Weiteren muss die RPID eine gültige RPID sein.

Eine RPID ist genau dann gültig, wenn sie mit der Domain des Origins der Relying Party übereinstimmt oder ein Suffix dieser ist.[1] Dabei ist sie unabhängig von dem Port des Origin. Besitzt die Relying Party beispielsweise den Origin “https://example.ascensus.com:18198”, dann sind “ascensus.com” und “example.ascensus.com” die einzigen gültigen RPIDs.

Es folgt die Ausführung der Erweiterungen. Dabei wird jede Erweiterung zunächst darauf überprüft, ob sie während einer Registrierung erlaubt ist. Falls eine Erweiterung weitere Interaktionen mit einem Authenticator benötigt, wird diese dem Authenticator übergeben.

Nun wird bei jedem erreichbaren Authenticator, der die Anforderungen aus den Optionen erfüllt, jeweils die *authenticatorMakeCredential* Operation mit folgenden Parametern aufgerufen:

clientDataHash ist der Hash der Client Data. Diese enthalten die Challenge, den Origin und gegebenenfalls den Identifikator des Token Binding, das in der Verbindung zur Relying Party genutzt wird. Token Binding [PNB18] ist ein Protokoll, um eine TLS Verbindung eindeutig zu identifizieren. Dadurch kann ein Cookie an eine bestimmte TLS Verbindung gebunden werden.

[1] Streng genommen muss die RPID entweder der *effectiveDomain* des Origins entsprechen oder eine *registrableDomainSuffix* der *effectiveDomain* des Origins sein. Das bedeutet, dass sie zunächst eine valide Domain sein muss, des Weiteren keine Top-Level Domain ist und entweder der Domain des Origin entspricht oder ein Suffix dieser ist. Falls keine Domain verfügbar ist, wird der Host an ihrer Stelle verwendet (siehe https://www.w3.org/TR/webauthn/#rp-id).

rp wird aus den Optionen übernommen.

user wird aus den Optionen übernommen.

requireResidentKey wird aus den Optionen übernommen.

userVerification ist genau dann "wahr", wenn Verifizierung des Nutzers laut der Relying Party durchgeführt werden muss oder wenn diese bevorzugt wird und der jeweilige Authenticator die Möglichkeit hierfür besitzt.

pubKeyCredParams wird aus den Optionen übernommen.

excludeCredentials wird aus den Optionen übernommen.

authenticatorExtensions beinhaltet die Erweiterungen, die zusätzliche Interaktionen mit dem Authenticator benötigt.

4. Authenticator erstellt Schlüsselpaar

Jeder aufgerufene Authenticator erhält die übergebenen Parameter und startet die *authenticatorMakeCredential* Operation, bei der zunächst ein neues Schlüsselpaar generiert wird.

Bevor der Nutzer seine Zustimmung geben kann, wird geprüft, ob mindestens eine in *excludeCredentials* enthaltenen Credential ID auf dem Authenticator registriert ist. "Registriert" bedeutet, dass die Credential ID entweder auf dem Authenticator gespeichert ist oder zu einem privaten Schlüssel entschlüsselt werden kann (siehe unten). Falls dies zutrifft und der Nutzer die Registrierung nicht abbricht, sondern fortführen möchte, wird dies dem WebAuthn Client explizit mitgeteilt und die Registrierung wird vom Authenticator abgebrochen. Dabei werden nur Credential IDs betrachtet, die im Kontext mit der RPID registriert wurden.

Um die Zustimmung für die Registrierung zu erhalten, sollten Informationen über die Relying Party und den zu registrierenden Nutzer angezeigt werden. Dabei muss eine Verifizierung des Nutzers bzw. das Testen der Anwesenheit einer Person in Abhängigkeit von *requireUserVerification* durchgeführt werden.

Nach Erhalten der Zustimmung wird ein aus einem öffentlichen und privaten Schlüssel bestehendes Schlüsselpaar zusammen mit einer Credential ID generiert. Hierfür wird die erste unterstützte Beschreibung in *pubKeyCredParams* genutzt. Es werden die folgenden Informationen auf dem Authenticator gespeichert:

id entspricht der generierten Credential ID.

privateKey entspricht dem privaten Schlüssel.

rpId entspricht der RPID.

userHandle entspricht der erhaltenen User Handle.

otherUI beinhaltet zusätzliche beliebige Informationen.

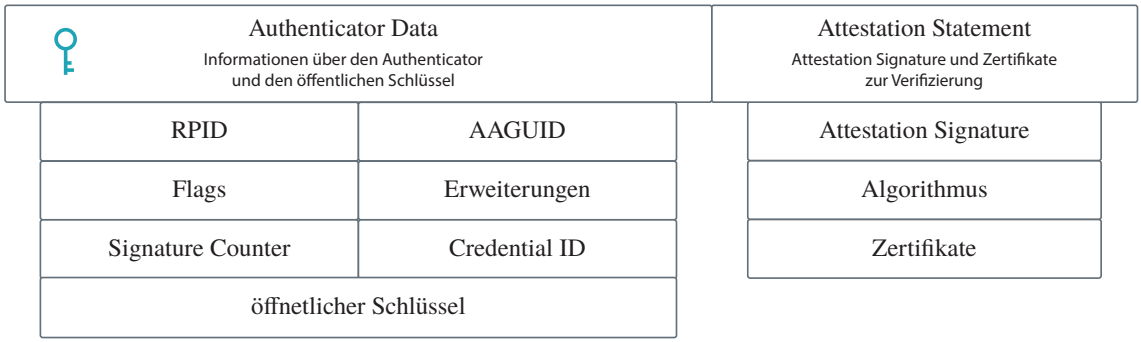


Abbildung 2.4: Attestation Object

Falls diese Informationen nicht gespeichert werden sollen, werden diese verschlüsselt und als Credential ID zurückgegeben. Dabei kann genau der Authenticator die Credential ID entschlüsseln, der diese auch verschlüsselt hat. Die Speicherung des User Handle ist optional.

Abschließend wird entweder ein *globaler* Signature Counter oder ein Signature Counter *pro Schlüsselpaar* initialisiert und es werden die unterstützen Erweiterungen ausgeführt. Falls der Authenticator keinen Signature Counter unterstützt, wird immer 0 zurückgegeben.

5. **Authenticator erstellt und sendet Antwort**

   In diesem Schritt sendet der Authenticator als Antwort für den WebAuthn Client ein Attestation Object, in dem der öffentliche Schlüssel und Informationen zur Verifizierung enthalten sind (siehe Abbildung 2.4).

   Ein Attestation Object besteht aus den Authenticator Data und einem Attestation Statement. Die Authenticator Data enthalten Informationen über das neu erstellte Schlüsselpaar, über den Authenticator, Angaben über durchgeführte Verifizierungen sowie weitere Informationen:

   rpidHash ist der Hash der RPID.

   flags enthält Markierungen für das erfolgreiche Testen der Anwesenheit einer Person bzw. der Verifizierung des Nutzers.

   signCount ist der Wert des Signature Counters.

   AAGUID enthält die AAGUID des Authenticators

   credentialID enthält die Credential ID des erstellten Schlüsselpaares.

   publicKey enthält den öffentlichen Schlüssel des erstellten Schlüsselpaares.

   extensions enthält die Ergebnisse der verarbeiteten Erweiterungen.

   Das Attestation Statement dient der Relying Party mithilfe der enthaltenen Attestation Signature zur Verifizierung und Überprüfung der Vertrauenswürdigkeit der Authenticator Data und besitzt einen Typ sowie Format. Dabei sind die zu signierenden Daten die Konkatenation von Authenticator Data und Client Data

   Der Typ bestimmt, welche Art von Schlüssel für die Signatur genutzt wird - der sogenannte Attestation Key. Dabei existieren folgende Möglichkeiten:

Basic Attestation Es wird ein privater Schlüssel des Authenticator genutzt. Dabei ist der Schlüssel bei Authenticatoren mit der gleichen AAGUID meistens der selbe.

Self Attestation Es wird der private Schlüssel des erstellten Schlüsselpaares genutzt.

Attestation CA Der Authenticator generiert ein asymmetrisches Schlüsselpaar zum Signieren und lässt dieses von einer Zertifizierungsstelle zertifizieren. Es kann für jede Signatur ein neues Schlüsselpaar generiert und genutzt werden.

ECDAA Der Authenticator nutzt *Elliptic Curve based Direct Anonymous Attestation* (ECDAA) zum Erzeugen von anonymen Signaturen [LCD18]. Das Verfahren basiert auf *Direct Anonymous Attestation*. Bei diesem ist es möglich, verifizierbare anonyme Signaturen zu erstellen, die keine Rückschlüsse auf den Nutzer haben und somit dessen Privatsphäre schützen, aber dennoch beweisen, dass der verwendete Authenticator vertrauenswürdig ist [GH18].

None Es wird kein Attestation Statement erzeugt.

Das Format gibt den Kontext an, in dem eine Attestation Signature generiert wurde, und bestimmt, beispielsweise durch Angabe von Zertifikaten, auf welche Weise das Attestation Statement verifiziert werden muss. In dieser Arbeit abstrahieren wir von diesem Konzept insoweit, dass wir eine Attestation Signature betrachten, die basierend auf ihrem Typ verifiziert wird.

6. WebAuthn Client sendet Antwort an Relying Party

Währenddessen wartet der WebAuthn Client auf die Antworten der Authenticatoren. Dabei kann der Nutzer die Registrierung über den WebAuthn Client oder an einem Authenticator abbrechen.

Falls ein Authenticator mitteilt, dass der Nutzer der Registrierung zugestimmt hat, obwohl mindestens eine Credential ID in *excludeCredentials* auf dem Authenticator gespeichert ist, wird dies der Relying Party sofort mitgeteilt. Ansonsten wird im Fall eines Abbruchs bis zum Auslaufen von *timeout* gewartet.

Falls ein Authenticator erfolgreich war, erhält der WebAuthn Client ein Attestation Object. Dieses sendet der WebAuthn Client zusammen mit den Client Data und den Ergebnissen der Erweiterungen an die Relying Party und bricht daraufhin bei allen weiteren Authenticatoren die Registrierung ab.

Zuvor wird gegebenenfalls das Attestation Statement des Attestation Objects geändert. Falls die Relying Party kein Attestation Statement fordert, wird das Attestation Statement nicht mitgesendet. Falls lediglich ein verifizierbares Attestation Statement gefordert ist, kann der WebAuthn Client die AAGUID und das Attestation Statement mit verifizierbaren Informationen ersetzen. Ansonsten muss das unveränderte Attestation Object übermittelt werden.

7. Relying Party verifiziert Antwort

Die Relying Party führt eine Verifizierung der erhaltenen Antwort durch, um letztendlich bei Erfolg den Nutzer zu registrieren. Dabei führt sie die im Folgenden beschriebenen Schritte durch. Falls ein Verifizierungsschritt fehlschlägt, wird die Registrierung abgebrochen.

Zunächst werden die in Client Data und Authenticator Data enthaltenen Informationen auf die zu erwartenden Werte überprüft:

challenge muss der generierten Challenge entsprechen.

origin muss dem Origin der Relying Party entsprechen.

tokenBinding muss dem Status und gegebenenfalls dem Identifikator des in der Verbindung genutzten Token Binding entsprechen.

flags müssen Markierungen für die Anwesenheit einer Person bzw. Verifizierung des Nutzers im Hinblick auf *userVerification* enthalten.

rpidHash muss dem Hash der RPID entsprechen.

Nun werden die Ergebnisse der Erweiterungen überprüft. Da die Durchführung von Erweiterungen optional ist, muss die Relying Party damit rechnen, dass die Erweiterungen keine Ergebnisse liefern. Dabei wird ebenfalls verifiziert, dass keine zusätzlichen Erweiterungen durchgeführt worden sind.

Als Nächstes wird das Attestation Statement mit dem Verifizierungsprozess des zugehörigen Typs verifiziert und anschließend auf Vertrauenswürdigkeit geprüft. Die Relying Party könnte beispielsweise *Self Attestation* oder eine bestimmte AAGUID als nicht vertrauenswürdig einstufen und die Registrierung abbrechen.

Nachdem überprüft wurde, dass die Credential ID nicht schon von einem anderen Nutzer verwendet wird, kann der zu registrierende Nutzer zusammen mit der Credential ID und dem öffentlichen Schlüssel abgespeichert werden.

Die Registrierung ist hiermit aus Sicht der Spezifikation erfolgreich abgeschlossen.

8. **Finale Antwort der Relying Party**

   Die Relying Party kann nun dem Nutzer mit zusätzlichen Informationen mitteilen, dass die Registrierung erfolgreich gewesen ist. Dieser Schritt ist nicht Teil der Spezifikation.

2.4 Authentifizierung

In diesem Abschnitt wird der Prozess einer Authentifizierung näher beschrieben. Eine Authentifizierung ist der Vorgang, bei dem ein Nutzer seine Identität gegenüber einer Relying Party mithilfe eines zuvor registrierten öffentlichen Schlüssels in Zusammenarbeit mit einem WebAuthn Client und einem Authenticator beweist.

Eine Übersicht der einzelnen Schritte ist in Abbildung 2.5 dargestellt. Dabei bestehen die einzelnen Schritte aus den in den vorherigen Kapiteln vorgestellten Operationen.

1. **Authentifizierungsanfrage**

   Der Prozess der Authentifizierung wird eingeleitet, indem ein Nutzer eine Authentifizierungsanfrage mithilfe seines WebAuthn Clients an eine Relying Party stellt. Teil der Anfrage kann ein Nutzername oder weitere Informationen sein, welche die Relying Party benötigt. Dieser Schritt wird in der Spezifikation nicht näher behandelt.

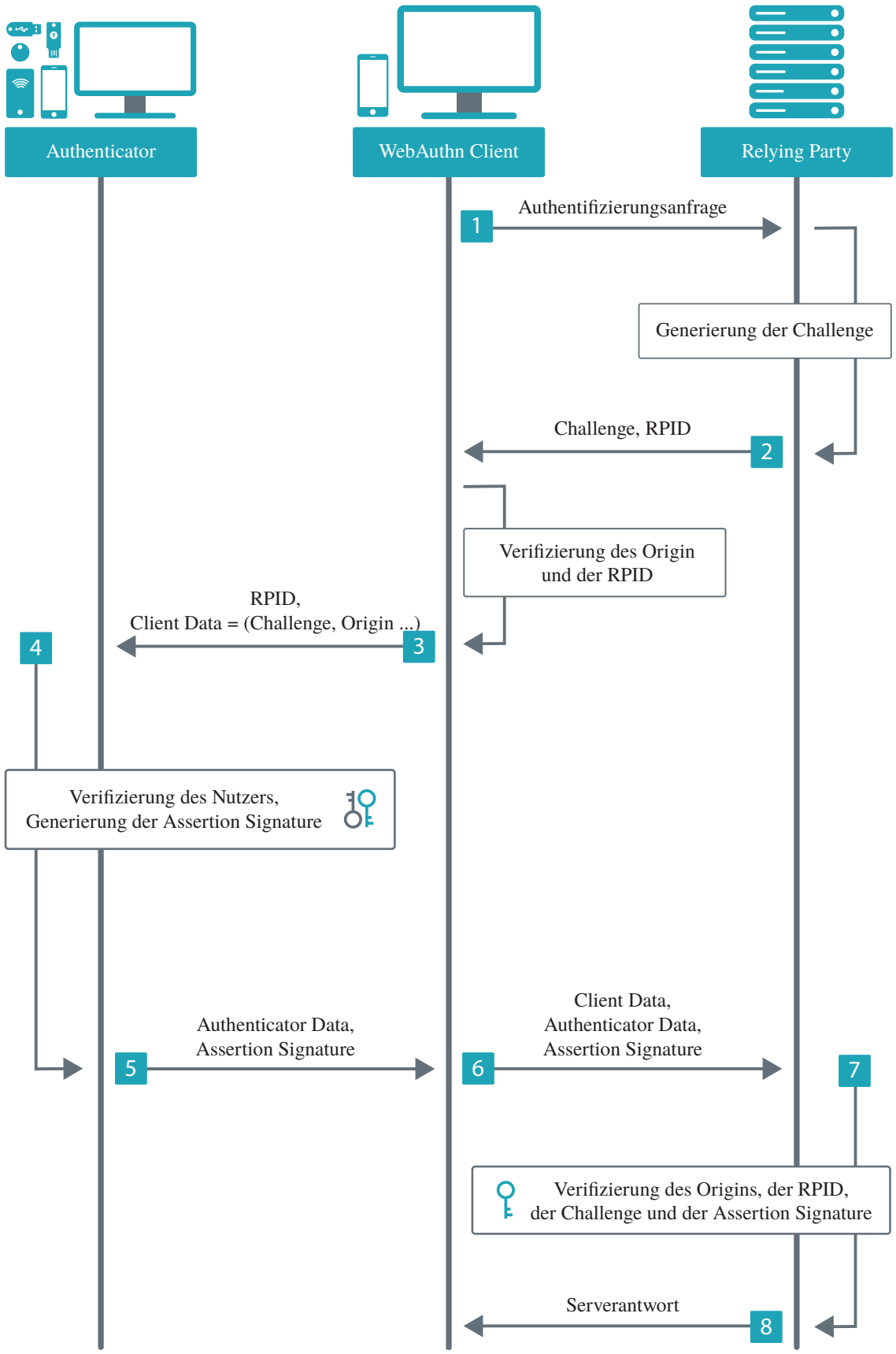


Abbildung 2.5: Einzelne Schritte während einer Authentifizierung

Abbildung 2.6: Assertion Signature

2. **Relying Party sendet Optionen**

   Die Relying Party erhält die Authentifizierungsanfrage und sendet folgende Optionen an den WebAuthn Client:

   challenge enthält eine neu generierte Challenge.

   timeout gibt an, nach welcher Zeit die Operation abgebrochen werden soll.

   rpid entspricht der RPID der Relying Party.

   allowCredentials enthält Credential IDs, die für die Authentifizierung genutzt werden dürfen. Diese Liste kann leer sein.

   userVerification bestimmt, ob eine Verifizierung des Nutzers auf dem Authenticator durchgeführt, bevorzugt oder verhindert werden muss.

   extensions ist eine Liste mit Erweiterungen, die zusätzlich während der Authentifizierung ausgeführt werden sollen.

3. **WebAuthn Client übergibt Optionen an Authenticator**

   Der WebAuthn Client erhält die oben beschriebenen Optionen und startet die *get* Operation, bei der zunächst eine Validierung und der Aufruf der Authenticatoren durchgeführt wird. Falls ein Verifizierungsschritt fehlschlägt, wird die Authentifizierung abgebrochen.

   Zunächst werden der Origin, die RPID und die Erweiterungen, wie während der Registrierung, überprüft bzw. ausgeführt.

   Daraufhin wird bei jedem erreichbaren Authenticator, der die Anforderungen aus den Optionen erfüllt, jeweils die *authenticatorGetAssertion* Operation mit folgenden Parametern aufgerufen:

   clientDataHash ist der Hash der Client Data. Diese enthalten, wie während der Registrierung, die Challenge, den Origin und gegebenenfalls den Identifikator des Token Binding, das in der Verbindung zur Relying Party genutzt wird.

   rpid enthält die RPID.

   userVerification ist genau dann “wahr”, wenn Verifizierung des Nutzers laut der Relying Party durchgeführt werden muss oder wenn diese bevorzugt wird und der jeweilige Authenticator die Möglichkeit hierfür besitzt.

   allowCredentials wird aus den Optionen übernommen.

   authenticatorExtensions beinhaltet die Erweiterungen, die zusätzliche Interaktion mit einem Authenticator benötigt.

4. **Authenticator erstellt Signatur**

   Ein Authenticator erhält die übergebenen Parameter und startet die *authenticatorGetAssertion* Operation, bei der zunächst der private Schlüssel des Nutzers ausgewählt werden muss, um daraufhin eine Assertion Signature zu erzeugen (siehe Abbildung 2.6).

   Der Authenticator listet alle im Kontext mit der RPID registrierten privaten Schlüssel auf. Sofern *allowCredentials* nicht leer ist, dürfen hierfür nur Schlüssel genutzt werden, deren Credential ID in *allowCredentials* aufgeführt wird. Falls kein Schlüssel den Kriterien entspricht, wird die Authentifizierung abgebrochen.

   Um die Zustimmung für die Authentifizierung zu erhalten, sollten Informationen über die Relying Party und den zu authentifizierenden Nutzer angezeigt werden. Dabei muss eine Verifizierung des Nutzers bzw. das Testen der Anwesenheit einer Person in Abhängigkeit von *requireUserVerification* durchgeführt werden.

   Nachdem der Nutzer einen Schlüssel ausgewählt hat, wird dieser genutzt, um die Assertion Signature zu generieren: eine Signatur über die Konkatenation von Authenticator Data und Client Data. Dabei besitzen die Authenticator Data folgende Parameter:

   rpidHash ist der Hash der RPID.

   flags enthält Markierungen für das erfolgreiche Testen der Anwesenheit einer Person bzw. der Verifizierung des Nutzers.

   signCount ist der Wert des Signature Counters.

   extensions enthält die Ergebnisse der verarbeiteten Erweiterungen.

   Zuvor wird entweder ein *globaler* Signature Counter oder ein Signature Counter *pro Schlüsselpaar* erhöht und es werden die unterstützen Erweiterungen ausgeführt. Falls der Authenticator keinen Signature Counter unterstützt, wird immer 0 zurückgegeben.

5. **Authenticator erstellt und sendet Antwort**

   In diesem Schritt sendet der Authenticator als Antwort für den WebAuthn Client die Credential ID des für die Signatur genutzten Schlüssels zusammen mit der Assertion Signature, den Authenticator Data und dem im Zusammenhang mit dem Credential ID gespeicherten User Handle.

6. **WebAuthn Client sendet Antwort an Relying Party**

   Währenddessen wartet der WebAuthn Client auf die Antworten der Authenticatoren.

   Falls der Nutzer oder ein Authenticator die Authentifzierung abbricht, wird bis zum Auslaufen von *timeout* gewartet, bevor die Relying Party benachrichtigt wird.

   Falls ein Authenticator erfolgreich war, sendet der WebAuthn Client die erhaltene Antwort zusammen mit den Client Data und den Ergebnissen der Erweiterungen an die Relying Party.

7. **Relying Party validiert Antwort**

   Die Relying Party führt eine Verifizierung der erhaltenen Antwort durch, um letztendlich bei Erfolg den Nutzer zu authentifizieren. Dabei führt sie die im Folgenden beschriebenen Schritte durch. Falls ein Verifizierungsschritt fehlschlägt, wird die Authentifizierung abgebrochen.

   Falls *allowCredentials* nicht leer ist, muss die erhaltene Credential ID in dieser Liste enthalten sein. Sofern in der Antwort ein User Handle enthalten ist, muss der erhaltene und der im Kontext der Credential ID gespeicherte User Handle übereinstimmen.

   Zunächst werden die in Client Data und Authenticator Data enthaltenen Informationen auf die zu erwartenden Werte überprüft:

   challenge muss der generierten Challenge entsprechen.

   origin muss dem Origin der Relying Party entsprechen.

   tokenBinding muss dem Status und gegebenenfalls dem Identifikator des in der Verbindung genutzten Token Binding entsprechen.

   flags müssen Markierungen für die Anwesenheit einer Person bzw. Verifizierung des Nutzers im Hinblick auf *userVerification* enthalten.

   rpidHash muss dem Hash der RPID entsprechen.

   Nun werden die Ergebnisse der Erweiterungen überprüft. Die Relying Party muss damit rechnen, dass die Erweiterungen keine Ergebnisse liefern. Dabei wird ebenfalls verifiziert, dass keine zusätzlichen Erweiterungen durchgeführt worden sind.

   Bei der Überprüfung des *signCount* muss die Relying Party selber entscheiden, ob nur ein streng monoton wachsender Signature Counter akzeptiert wird.

   Als Nächstes wird mit dem im Zusammenhang mit der Credential ID gespeicherten öffentlichen Schlüssel verifiziert, dass die Assertion Signature eine gültige Signatur über die Konkatenation von Authenticator Data und Client Data ist.

   Die Authentifizierung ist nach dem Aktualisieren des Signature Counters aus Sicht der Spezifikation erfolgreich abgeschlossen.

8. **Finale Antwort der Relying Party**

   Die Relying Party kann nun dem Nutzer mit zusätzlichen Informationen mitteilen, dass die Authentifizierung erfolgreich gewesen ist. Dieser Schritt ist nicht Teil der Spezifikation.

2.5 Erweiterungen

Während einer Registrierung oder einer Authentifizierung kann eine Relying Party zusätzlich Operationen in Form von Erweiterungen durchführen lassen (siehe Abbildung 2.7). Diese sollen durch zusätzliche Informationen dazu beitragen, die Anforderungen der Relying Party zu erfüllen, die nicht über die ursprünglichen Optionen von WebAuthn ermöglicht werden. Dabei muss damit gerechnet werden, dass die Erweiterungen von dem WebAuthn Client oder Authenticator ignoriert und nicht ausgeführt werden. Folgende Erweiterungen stehen zur Auswahl:

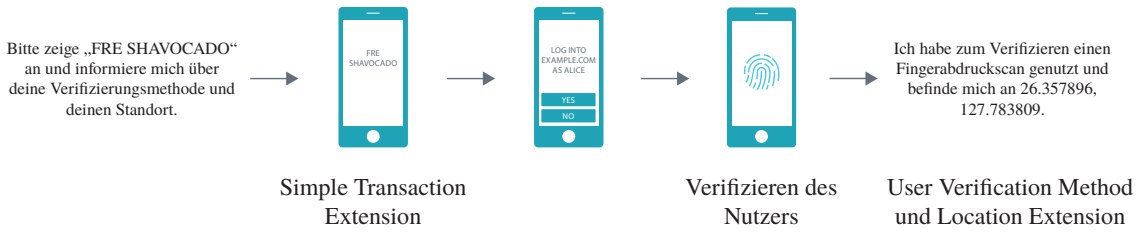


Abbildung 2.7: Beispiel für Erweiterungen

FIDO AppID Private Schlüssel, die während der Registrierung bei FIDO U2F generiert werden, verwenden einen anderen Identifikator für die Relying Party. Daher wird anstelle der RPID die AppID [LHBB18] an den Authenticator übergeben. Dadurch können bereits existierende U2F Schlüssel weiterhin genutzt werden.

Simple Transaction Es wird ein Text übergeben, der auf dem Authenticator vor der Verifizierung des Nutzers bzw. Test der Anwesenheit einer Person angezeigt werden soll.

Generic Transaction Es können nicht nur Texte, sondern auch Bilder übergeben werden, die auf dem Authenticator vor der Verifizierung des Nutzers bzw. Test der Anwesenheit einer Person angezeigt werden soll.

Authenticator Selection Es werden die AAGUIDs der Authenticatoren übergeben, die während einer Registrierung erlaubt sind.

Supported Extensions Es wird angefragt, welche Erweiterungen der Authenticator unterstützt.

User Verification Index Es wird der User Verification Index angefragt, der die vom Authenticator für die Verifizierung genutzten Daten identifiziert. Mit diesem kann die Relying Party feststellen, ob die gleiche Person wie bei vorherigen Operationen im Zusammenhang mit dem zu authentifizierenden Nutzer auf dem Authenticator verifiziert wurde.

User Verification Method Es wird angefragt, mit welcher Methode der Authenticator eine Verifizierung des Nutzers durchgeführt hat.

Location Es wird die geographische Position des Authenticators angefragt.

Biometric Authenticator Performance Bounds Es sind nur Authenticatoren erlaubt, deren biometrische Verifizierung innerhalb der definierten Grenzen für Falschakzeptanzrate und Falscherkennungsrate liegt.

2.6 Schlüssel-Widerruf

In diesem Abschnitt werden Szenarien behandelt, in denen registrierte Schlüssel widerrufen werden sollen. Die Folge eines Schlüssel-Widerrufs ist die Außerbetriebnahme des öffentlichen und privaten Schlüssels. Diese werden nach einem Schlüssel-Widerruf für keine weiteren Operationen mehr angeboten bzw. verwendet.

Abbildung 2.8: Schlüssel-Widerruf eines verlorenen Schlüssels

Der Schlüssel-Widerruf wird bis auf das Nennen von Beispielen in der Spezifikation nicht näher behandelt. Das liegt daran, dass sie Aufgabe der Relying Party ist und somit nicht Teil der WebAuthentication API. Es bleibt daher einer Relying Party überlassen, wie diese einen Schlüssel-Widerruf handhabt.

Folgende Szenarien sind Beispiele eines Schlüssel-Widerrufs:

Verlorener privater Schlüssel Für den Fall, dass ein Nutzer seinen privaten Schlüssel verliert, bietet die Relying Party die Möglichkeit an, diesen als verloren zu markieren (siehe Abbildung 2.8). Die Handhabung ist vergleichbar mit dem Verlust eines Passwortes. Der als verloren gemeldete Schlüssel wird außer Betrieb gesetzt und somit nicht mehr für Authentifizierungen angeboten. Ein Schlüssel kann verloren gehen, wenn beispielsweise der zugehörige Authenticator gestohlen, zerstört oder zurückgesetzt wurde.

Inaktivität eines Schlüssels Nachdem ein öffentlicher Schlüssel für einen längeren Zeitraum nicht mehr genutzt wurde, wird dieser außer Dienst genommen. Dieser Fall tritt beispielsweise ein, wenn ein Nutzer einen weiteren öffentlichen Schlüssel registriert hat und ausschließlich diesen während einer Authentifizierung verwendet.

Schlüssel-Widerruf Entscheidet sich ein Nutzer dafür, einen öffentlichen Schlüssel nicht mehr zu nutzen, hat er die Möglichkeit, diesen auf dem zugehörigen Authenticator zu widerrufen. Der öffentliche Schlüssel muss nicht explizit auf der Relying Party widerrufen werden, da dieser aufgrund von Inaktivität nach einiger Zeit außer Betrieb genommen wird.

3 Implementierung

In diesem Kapitel wird der praktische Teil dieser Arbeit vorgestellt: Eine Demo-Implementierung einer Relying Party, die WebAuthn nutzt, um Nutzer zu registrieren und zu authentifizieren. Die Implementierung besteht aus zwei Teilen: Der erste ist die Umsetzung der Relying Party, der zweite die Nutzung der WebAuthentication API mithilfe des WebAuthn Clients. Hierfür entwerfen wir in Form eines Node.js Servers die Relying Party "Ascensus".

Im Folgenden wird die Implementierung im Bezug auf Umsetzung und Funktionalität sowie Abhängigkeiten zusammen mit Beispielen vorgestellt. Dabei liegt der Fokus der Implementierung auf der Verwendung von WebAuthn.

Der Quellcode ist sowohl auf der beigelegten CD als auch online in meinem GitHub Repository [Stö18c] zu finden.

3.1 Parteien

In diesem Abschnitt betrachten wir die technische Realisierung der Relying Party und stellen den zu verwendenden WebAuthn Client und Authenticator vor.

3.1.1 Relying Party

Um den Server mit dem Origin "https://ascensus.com" von Ascensus zu erstellen, entscheiden wir uns für einen Node.js Server mit dem Express Framework, Node.js v8.9.4 und ECMAScript 6. Der Server besitzt eine Verbindung zu einer lokalen nicht persistenten MongoDB Datenbank im Arbeitsspeicher. Diese wird zum Speichern von Account- und Sitzungsinformationen genutzt. Jeder Besucher der Website besitzt eine eigene Sitzung, deren Identifikator als Cookie gespeichert wird. Der signierte Cookie kann nur bei einer durch HTTPS gesicherten Verbindung verwendet werden. Falls ein Nutzer authentifiziert ist, wird dieser persistent in der Sitzung gespeichert.

Da die WebAuthentication API nur bei einer durch HTTPS gesicherten Verbindung zur Verfügung steht, ist Ascensus über TLS mit einem selbst signierten Zertifikat gesichert. Die Verbindung nutzt kein Token Binding. Zusätzlich besitzt jede POST Route einen CSRF Schutz. Eine Übersicht der Routen und damit der Funktionalität ist in Tabelle 3.1 dargestellt.

Um den Server zu starten, sollte der Befehl `npm install && node ./bin/www` genutzt werden. Beim erstmaligen Starten wird automatisch die letzte Version von MongoDB heruntergeladen und installiert.

Um den Origin des Servers nutzen zu können, kann unter Windows der Eintrag `127.0.0.1 ascensus.com` in der Datei `C:/Windows/System32/drivers/etc/hosts` hinzugefügt werden.

Route	Beschreibung
`GET /`	Startseite, um sich zu registrieren oder authentifizieren
`GET /profile`	Profil eines Nutzers mit der Möglichkeit einen zusätzlichen Authenticator zu registrieren
`GET /user`	Registrierte Informationen eines Nutzers
`POST /register`	Erster Teil einer Registrierung: Überprüfung des Nutzernamens, Bereitstellen der Optionen (inkls. einem User Handle und einer Challenge), Aufruf der WebAuthentication API
`POST /register/callback`	Zweiter Teil einer Registrierung: Verifizierung des erhaltenen Attestation Objects und registrieren des öffentlichen Schlüssel des Nutzers
`POST /authenticate`	Erster Teil einer Authentifizierung: Heraussuchen des Nutzers über einen Nutzernamen, Bereitstellen der Optionen (inkls. der Credential ID des gefundenen Nutzers und einer Challenge), Aufruf der WebAuthentication API
`POST /authenticate/callback`	Zweiter Teil einer Authentifizierung: Verifizierung der erhaltenen Assertion Signature und Authentifizieren des Nutzers
`POST /logout`	Ausloggen des authentifizierten Nutzers
`GET /help`	Hilfeseite

Tabelle 3.1: Übersicht der Routen

Während einer Registrierung oder Authentifizierung werden alle gesendeten Nachrichten und Überprüfungen in der Konsole des Servers und des Browsers angezeigt.

3.1.2 WebAuthn Client

Als WebAuthn Client nutzen wir den Browser Firefox Quantum 61.0.1 zusammen mit Microsoft Windows 10 und ECMAScript 6. Dieser Browser implementiert bereits die WebAuthentication API und übernimmt die Kommunikation mit dem Authenticator.

Die WebAuthentication API stellt folgende Operationen bereit, die von der Relying Party durch ein Skript aufgerufen werden können:

- `navigator.credentials.create()` für eine Registrierung
- `navigator.credentials.get()` für eine Authentifizierung

3.1.3 Authenticator

Die Demo ist auf einen FIDO U2F Security Key als Authenticator ausgelegt. Der Authenticator muss aufgrund des genutzten Browsers nicht implementiert oder gekauft werden: Während der Entwicklung wurde ein in Firefox enthaltener Softtoken als Authenticator genutzt. Dieser kann wie folgt aktiviert werden:

1. Öffne im Browser `about:config`

Listing 3.1 Optionen während Registrierung

```
let RegistrationOptions = {
  rp: {
    name: 'ascensus',
    id: 'ascensus.com' // RPID
  },
  user: {
    id: utils.userHandle(), // 32 random bytes
    name: name,
    displayName: name
  },
  challenge: utils.challenge(), // 32 random bytes
  pubKeyCredParams: [{
    type: "public-key",
    alg: -7
  }], // -7 for "ES256" as registered in the IANA COSE Algorithms registry
  timeout: 60000,
  excludeCredentials: [], // Nothing to exclude
  authenticatorSelection: {
    authenticatorAttachment: 'cross-platform',
    requireResidentKey: false,
    userVerification: 'preferred' // User Verification not required
  },
  attestation: "none", // No Attestation Object requested
  extensions: {} // No extensions requested
};
```

2. Setze den Wert `security.webauth.webauthn` auf `true`
3. Setze den Wert `security.webauth.webauthn_enable_softtoken` auf `true`
4. Setze den Wert `security.webauth.webauthn_enable_usbtoken` auf `false`

Ein Authenticator, der das U2F Protokoll nutzt, verwendet einen *globalen* Signature Counter und hat keine Möglichkeit einen User Handle zu speichern [BCH18].

Um die Einsatzfähigkeit der Demo in der Praxis zu zeigen, wurde diese nach der Entwicklung erfolgreich mit einem Yubikey 4 in Kombination mit Windows 10 und Mozilla Firefox Quantum 61.0.1 sowie Goolge Chrome (Version 69.0.3497.100) getestet.

3.2 Registrierung

Ein Nutzer, der sich bei Ascensus registrieren möchte, besucht die Startseite der Internetseite. Auf dieser kann der Nutzer über ein Formular einen Nutzernamen per `POST` Anfrage an die Route `/register` des Servers senden (siehe Abbildung 3.1). Serverseitig wird überprüft, ob der Nutzername bereits in der Datenbank vorhanden ist. Falls dies der Fall ist, wird die Registrierung abgebrochen. Ansonsten werden folgende Optionen generiert und in der Sitzung des Nutzers gespeichert.

Die Optionen sind aus einem Beispiel der Spezifikation entnommen und auf unsere Implementierung angepasst. Sie enthalten die Standardeinstellungen (siehe Listing 3.1): Diese beinhalten die RPID "ascensus.com", eine neu generierte 32 Byte lange zufällige Bitfolge als User Handle, den erhaltenen Nutzernamen und eine neu generierte 32 Byte lange zufällige Bitfolge als Challenge. Zum Generieren des Schlüsselpaares soll der Algorithmus ES256 eingesetzt werden. Der zugehörige private Schlüssel muss nicht auf dem Authenticator gespeichert werden. Des Weiteren wird eine Verifizierung des Nutzers bevorzugt, es wird kein Attestation Statement gefordert und es werden keine Erweiterungen ausgeführt.

Falls der Nutzer bereits authentifiziert ist und dieser somit einen weiteren öffentlichen Schlüssel registrieren möchte, werden sowohl die Informationen des authentifizierten Nutzers in `user` als auch seine bereits registrierte Credential ID in `excludeCredentials` übergeben.

Der Nutzer wird daraufhin auf eine Seite weitergeleitet, die ihn dazu auffordert, die Registrierung auf einem Authenticator fortzuführen bzw. zu autorisieren (siehe Abbildung 3.2). Im Hintergrund wird folgendes Skript vom Browser ausgeführt (siehe Listing 3.2):

1. Zunächst wird überprüft, ob die WebAuthentication API zur Verfügung steht und damit gleichzeitig, ob die Verbindung über `https` gesichert ist.
2. Als Nächstes müssen die `Buffer` der Challenge und des User Handle jeweils in ein `Uint8Array` konvertiert werden.
3. Es folgt die Verwendung der WebAuthentication API: `navigator.credentials.create()` wird mit den erhaltenen Optionen aufgerufen.
4. Sobald der Nutzer die Registrierung am Authenticator autorisiert und erfolgreich abschließt, gibt die WebAuthentication API als Antwort das vom Authenticator erhaltene Attestation Object zusammen mit den Client Data zurück. Diese Antwort wird zunächst als `JSON` kodiert.
5. Anschließend wird die Antwort über ein nicht sichtbares Formular per `POST` Anfrage an die Route `/register/callback` des Servers gesendet.

Wie in Kapitel 2.3 beschrieben, wird die Antwort serverseitig verifiziert. Dabei wird keine Attestation Signature erwartet. Falls eine Überprüfung fehlschlägt, wird die Registrierung mit einer Fehlermeldung über die jeweilige fehlgeschlagene Überprüfung abgebrochen. Für die Verifizierung müssen zunächst Informationen, wie die Challenge, aus den Sitzungsdaten geladen werden und die erhaltene Antwort muss decodiert werden. Die ersten 32 Byte der `authData` enthalten beispielsweise den `SHA256` Hash der RPID, die Bytes 55 bis Ende den als `COSE` codierten öffentlichen Schlüssel.

Bei einer erfolgreichen Verifizierung wird ein neuer Nutzer in der Datenbank mit folgenden Informationen gespeichert:

userHandle entspricht dem User Handle aus den Sitzungsdaten.

name, displayName entspricht dem Nutzernamen aus den Sitzungsdaten.

credentialID entspricht der erhaltenen Credential ID.

publicKey enstpricht dem erhaltenen öffentlichen Schlüssel.

signCounter entspricht dem Wert des erhaltenen Signature Counters.

Der Nutzer wird daraufhin auf die Startseite weitergeleitet, auf der er sich authentifizieren kann.

Falls der Nutzer bereits authentifiziert ist, wird der zusätzliche Schlüssel in unserer Demo nicht gespeichert.

3.3 Authentifizierung

Ein Nutzer, der sich bei Ascensus authentifizieren möchte, besucht die Startseite der Internetseite. Auf dieser kann der Nutzer über ein Formular einen Nutzernamen per `POST` Anfrage an die Route `/authenticate` des Servers senden (siehe Abbildung 3.1). Serverseitig wird in der Datenbank anhand des Nutzernamens nach dem zu authentifizierenden Nutzer gesucht. Falls kein Nutzer gefunden wird, wird die Registrierung abgebrochen. Ansonsten werden folgende Optionen generiert und in der Sitzung des Nutzers gespeichert.

Die Optionen sind aus einem Beispiel der Spezifikation entnommen und auf unsere Implementierung angepasst. Sie enthalten die Standardeinstellungen (siehe Listing 3.3): Diese beinhalten die RPID “ascensus.com”, eine neu generierte 32 Byte lange zufällige Bitfolge als Challenge und in *allowCredentials* die Credential ID des gefundenen Nutzers. Diese übertragen wir nicht, wie in der Datenbank gespeichert, als `base64` codiert, sondern als `Buffer`. Des Weiteren wird eine Verifizierung des Nutzers bevorzugt und es werden keine Erweiterungen ausgeführt.

Wir verwenden die *allowCredentials* Option, da der Authenticator des Nutzers das U2F Protokoll verwendet. Bei diesem muss eine Credential ID während einer Authentifizierung übergeben werden [BBL17].

Der Nutzer wird daraufhin auf eine Seite weitergeleitet, die ihn dazu auffordert, die Authentifizierung auf einem Authenticator fortzuführen bzw. zu autorisieren (siehe Abbildung 3.2). Im Hintergrund wird ein Skript vom Browser mit der gleichen Struktur wie während der Registrierung ausgeführt:

1. Zunächst wird überprüft, ob die WebAuthentication API zur Verfügung steht und damit gleichzeitig, ob die Verbindung über `https` gesichert ist.
2. Als Nächstes werden die `Buffer` der Challenge und der Credential ID jeweils zu einem `Uint8Array` konvertiert.
3. Es folgt die Verwendung der WebAuthentication API: `navigator.credentials.get()` wird mit den erhaltenen Optionen aufgerufen.
4. Sobald der Nutzer die Authentifizierung am Authenticator autorisiert und erfolgreich abschließt, gibt die WebAuthentication API als Antwort die vom Authenticator erhaltenen Authenticator Data und Assertion Signature zusammen mit den Client Data zurück. Diese Antwort wird zunächst als `JSON` codiert.
5. Anschließend wird die Antwort über ein nicht sichtbares Formular per `POST` Anfrage an die Route `/authenticate/callback` des Servers gesendet.

Wie in Kapitel 2.4 beschrieben, wird die erhaltene Antwort serverseitig verifiziert. Für die Verifizierung müssen zunächst Informationen, wie die Challenge, aus den Sitzungsdaten geladen werden und die erhalten Antwort muss decodiert werden.

Listing 3.2 Verwendung der WebAuthentication API während einer Registrierung

```
"use strict";

// Step 1: Check if WebAuthentication API is available (and therefore if connection is secure)
if (!window.PublicKeyCredential) {
    showError('Web Authentication API not found');
}

// Step 2: Encode challenge, user.id and each id in excludeCredentials as Uint8Array
let options = <%- JSON.stringify(RegistrationOptions) %>;
options.challenge = new Uint8Array(options.challenge.data);
options.user.id = new Uint8Array(options.user.id.data);

for (let i = 0; i < options.excludeCredentials.length; i++) {
    options.excludeCredentials[i].id = new Uint8Array(options.excludeCredentials[i].id.data);
}

// Step 3: Call the registration on the authenticator using WebAuthentication API
navigator.credentials.create({"publicKey": options})
    .then(function (PublicKeyCredential) {

        // Step 4: Encode the received answer as JSON
        let attestationObject = CBOR.decode(PublicKeyCredential.response.attestationObject);
        let PublicKeyCredentialJSON = {
            id: PublicKeyCredential.id,
            rawId: Array.from(new Uint8Array(PublicKeyCredential.rawId)),
            response: {
                attestationObject: {
                    attStmt: {},
                    authData: Array.from(attestationObject.authData),
                    fmt: attestationObject.fmt
                },
                clientDataJSON: Array.from(new Uint8Array(
                                        PublicKeyCredential.response.clientDataJSON))
            },
            type: PublicKeyCredential.type
        }

        // Step 5: Send the answer to the server
        let data = JSON.stringify(PublicKeyCredentialJSON);
        document.getElementById('PublicKeyCredentialInput').value = data;
        document.getElementById('form').submit();

    }).catch(function (err) {
    // No acceptable authenticator or user refused consent
});
```

Listing 3.3 Optionen während Authentifizierung

```
let AuthenticationOptions = {
  challenge: utils.challenge(), // 32 random bytes
  timeout: 60000,
  rpId: 'ascensus.com', // RPID
  allowCredentials: [{type: "public-key",
            id: Buffer.from(user.credentialId, 'base64'), // User's Credential ID
            transports: ["usb", "nfc", "ble"]}],
  userVerification: "preferred",
  extensions: {} // No extensions required
};
```

Bei einer erfolgreichen Verifizierung ist der Nutzer, der in der Datenbank im Kontext der erhaltenen Credential ID abgespeichert ist, erfolgreich authentifiziert und wird auf sein Profil weitergeleitet. Wir identifizieren den authentifizierten Nutzer über die Credential ID, da der eingesetzte Authenticator keinen User Handle in seiner Antwort zurückgibt. Falls eine Überprüfung fehlschlägt, wird die Authentifizierung mit einer Fehlermeldung über die jeweilige fehlgeschlagene Überprüfung abgebrochen.

In unserer Implementierung kann das in Kapitel 4.2.1.10 vorgestellte Problem nicht auftreten, da wir den zu authentifizierenden Nutzer über die Credential ID identifizieren.

3.4 Szenarien

In diesem Abschnitt betrachten wir aus der Sicht des Nutzers mögliche Szenarien der Demo beim Verwenden von WebAuthn. Dabei verwendet der Nutzer die oben beschriebenen Parteien.

Erfolgreiche Registrierung Alice besucht die Startseite von Ascensus (siehe Abbildung 3.1), gibt ihren Nutzernamen ein und stellt eine Registrierungsanfrage. Sie wird aufgefordert den nächsten Schritt auf ihrem FIDO U2F Security Key fortzuführen (siehe Abbildung 3.2). Auf diesem bestätigt sie die Anfrage, indem sie einen Knopf betätigt. Alice erhält daraufhin am Computer die Bestätigung, dass sie sich erfolgreich registriert hat (siehe Abbildung 3.3).

Erfolgreiche Authentifizierung Alice möchte sich in ihrem Account einloggen. Dafür besucht sie erneut die Startseite, gibt ihren Nutzernamen ein und stellt eine Authentifizierungsanfrage. Nach der Sie wird aufgefordert den nächsten Schritt auf ihrem FIDO U2F Security Key fortzuführen. Auf diesem bestätigt sie die Anfrage, indem sie einen Knopf betätigt. Alice erhält daraufhin am Computer die Bestätigung, dass sie sich erfolgreich authentifiziert hat und wird auf ihr Nutzerprofil weitergeleitet (siehe Abbildung 3.4).

Weiteres Gerät registrieren Alice möchte einen weiteren FIDO U2F Security Key registrieren. Dafür loggt sie sich mit dem ursprünglichen Key ein und startet eine Registrierung eines weiteren Gerätes. Während der Registrierung wird in *excludeCredentials* die bereits registrierte Credential ID übergeben, sodass nicht mehrere Credential IDs des gleichen Accounts auf einem Authenticator registriert ist. Sofern die Registrierung erfolgreich war, wird diese Alice mitgeteilt (siehe Abbildung 3.5).

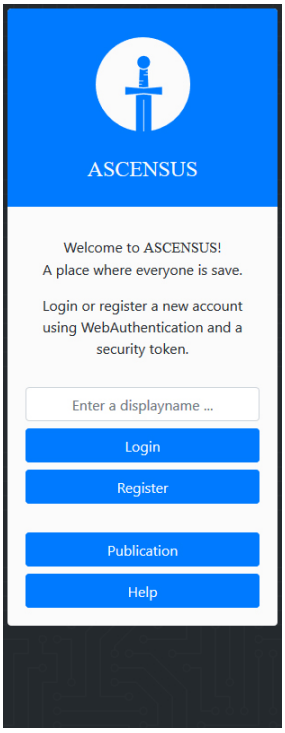


Abbildung 3.1: Startseite

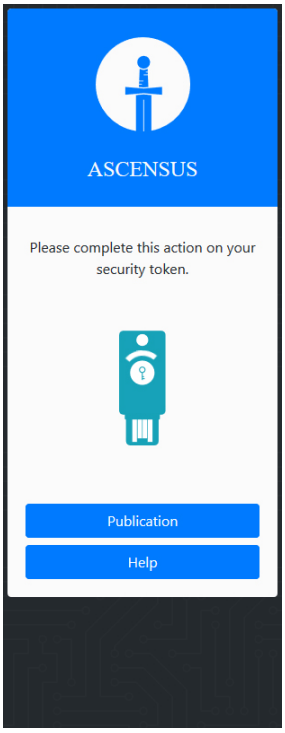


Abbildung 3.2: Aufforderung an Authenticator

Kein sicherer Kontext Alice möchte sich registrieren. Dabei wird keine über `https` gesicherte Verbindung genutzt. Bei dem Versuch sich zu registrieren, erhält sie eine Fehlermeldung (siehe Abbildung 3.6).

excludeCredentials Alice möchte einen weiteren FIDO U2F Security Key registrieren. Dafür loggt sie sich mit dem ursprünglichen ein und startet eine Registrierung eines weiteren Gerätes. Hierfür verwendet sie allerdings den gleichen Authenticator. Daher wird die Registrierung abgebrochen (siehe Abbildung 3.5).

Fehlgeschlagene Authentifizierung Eve versucht sich in Alice Account einzuloggen. Da sie allerdings Alice privaten Schlüssel nicht besitzt, muss sie die Attestation Signature raten. Die serverseitige Verifizierung schlägt fehl und eine entsprechende Fehlermeldung wird angezeigt (siehe Abbildung 3.8).

Phishing Angriff Alice möchte sich bei Eves Website einloggen. Währenddessen versucht Eve sich bei einer Relying Party als Alice ausgeben und startet bei dieser eine Authentifizierung in Alice Namen. Die erhaltene Challenge leitet sie an Alice weiter und erhält daraufhin eine von Alice erzeugte Attestation Signature. Diese leitet sie an die Relying Party weiter. Da Alice die Signatur allerdings mit dem Origin von Eves Website erzeugt hat, schlägt Eves Angriff fehl (siehe Abbildung 3.9).

Replay Angriff Eve hat eine Attestation Signature aus einer früheren Authentifizierung von Alice in ihrem Besitz und sendet diese nun an die Relying Party, um sich als Alice auszugeben. Da die Challenge allerdings nicht mit der für diese Authentifizierung neu generierten Challenge übereinstimmt, wird die Authentifizierung abgebrochen (siehe Abbildung 3.10).

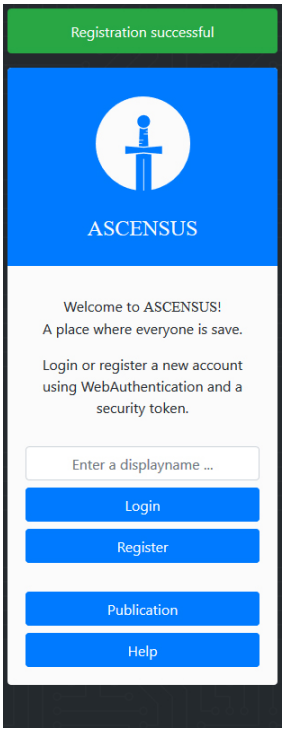


Abbildung 3.3: Erfolgreiche Registrierung

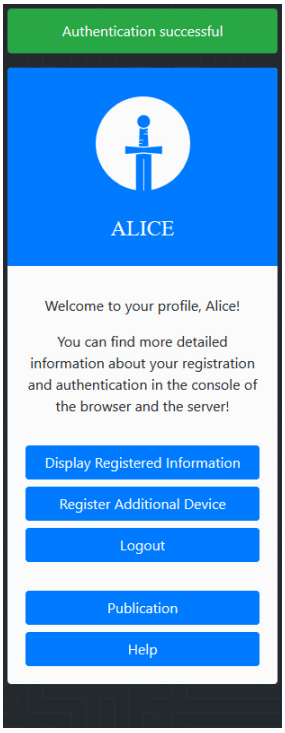


Abbildung 3.4: Erfolgreiche Authentifizierung

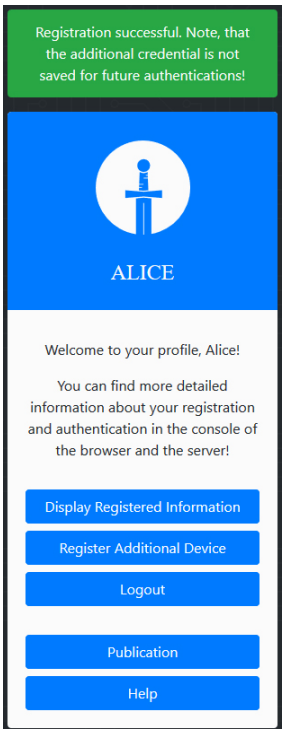


Abbildung 3.5: Weiteres Gerät registrieren

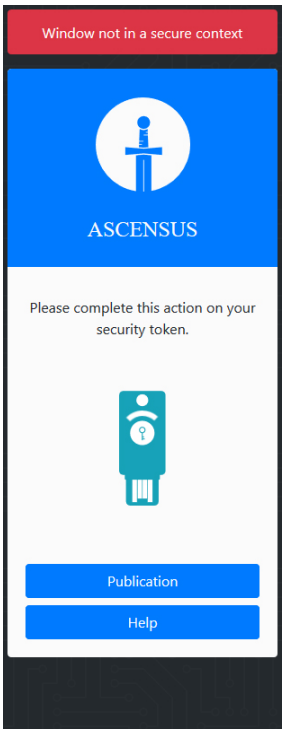


Abbildung 3.6: Kein sicherer Kontext

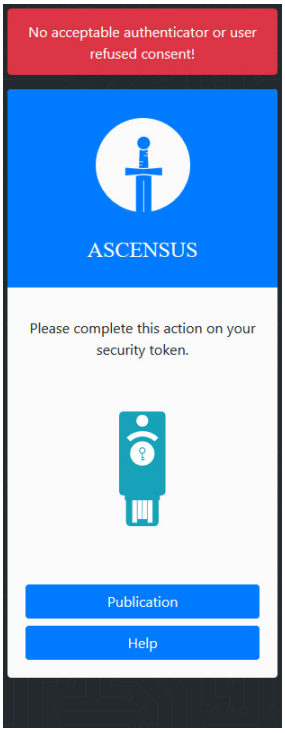


Abbildung 3.7: Credential ID bereits registriert

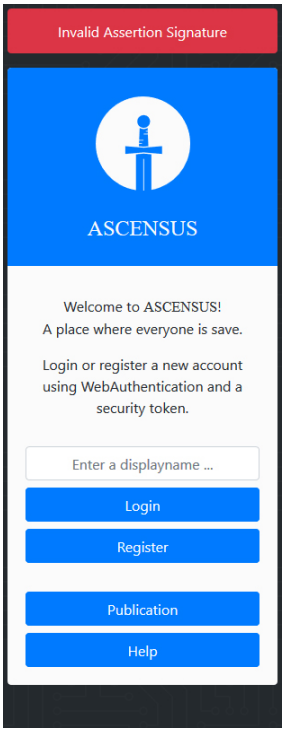


Abbildung 3.8: Ungültige Assertion Signature

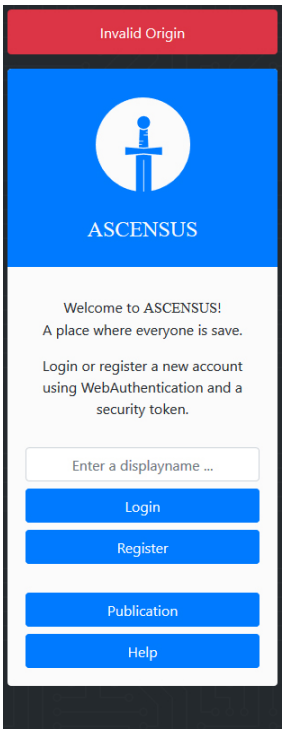


Abbildung 3.9: Ungültiger Origin

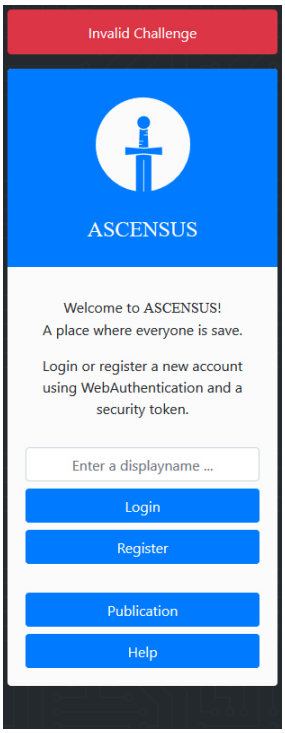


Abbildung 3.10: Ungültige Challenge

4 Sicherheitsanalyse

In diesem Kapitel befassen wir uns mit einer informellen Sicherheitsanalyse von WebAuthn. Hierfür legen wir zunächst verschiedene Anforderungen an das Protokoll in Form von Sicherheitszielen fest, treffen für die Analyse geltende Annahmen und beschreiben die Bedrohungslage.

Wir untersuchen die Sicherheit, indem wir aufführen, wieso die während einer Registrierung bzw. Authentifizierung durchzuführenden Überprüfungen der einzelnen Parteien notwendig sind und welche Auswirkungen das Verwenden verschiedener Optionen von WebAuthn auf die Sicherheitsziele hat. Des Weiteren untersuchen wir die Kommunikationskanäle im Hinblick auf einen Man-in-the-Middle und welche Gefahr von diesem ausgeht. Anschließend diskutieren wir aufgrund der Analyse aus, welche Sicherheitsziele gewährleistet werden können.

Während der Sicherheitsanalyse betrachten wir in Kapitel 4.2.1.10 einen Angriff auf das Authentifizierungsverfahren und widerlegen in Kapitel 4.2.2.2 die in der Spezifikation aufgeführte Behauptung, dass WebAuthentication während einer Registrierung resistent gegenüber einem Man-in-the-Middle sei, der den sicheren Kontext ignorieren kann. Beide Probleme wurden von den Autoren der Spezifikation anerkannt. Für den Angriff auf das Authentifizierungsverfahren wurde bereits eine Lösung gefunden.

4.1 Sicherheitsziele und Annahmen

Im Folgenden definieren wir verschiedene Sicherheitsziele und Annahmen im Hinblick auf WebAuthn und mögliche Angreifer. Wir bezeichnen den Nutzer, der bei einer Relying Party einen Account registriert hat, als den Besitzer des registrierten Accounts. Dabei kann ein Nutzer mehrere Accounts bei verschiedenen Relying Parties besitzen. Als Grundlage für die Sicherheitsziele verwenden wir die FIDO Security Reference [Lin18]:

Authentication Ein Nutzer kann sich nur in die vom ihm registrierten Accounts bei einer konformen Relying Party einloggen. Insbesondere ist es einem Angreifer nicht möglich, sich als ein anderer Nutzer auszugeben.

Privacy Ein Nutzer teilt während einer Registrierung oder Authentifizierung nur so viele persönliche Informationen über sich mit wie nötig. Insbesondere ist es nicht möglich, einen Nutzer, der einen Account bei einer Relying Party besitzt, als der Besitzer eines weiteren Accounts bei der gleichen oder einer weiteren Relying Party zu identifizieren.

User Consent Eine erfolgreiche Registrierung oder Authentifizierung kann nicht ohne die Zustimmung eines Nutzers durchgeführt werden. Dabei muss dem Nutzer der Kontext bewusst sein. Das bedeutet, dass der Nutzer unter Kenntnis der RPID weiß, ob durch seine Zustimmung eine Registrierung oder Authentifizierung autorisiert wird und, im Falle einer Authentifizierung, für welchen Account er sich authentifiziert.

Registration Verifiability Die Relying Party hat die Möglichkeit, die Integrität und Authentizität der Informationen während einer Registrierung zu verifizieren. Insbesondere kann sie das Modell des Authenticators sowie nötige Zertifikate für eine Verifizierung abrufen.

Wir treffen folgende Annahmen. Sobald eine Annahme nicht mehr zutrifft, können die oben erwähnten Sicherheitsziele nicht gewährleistet werden:

Attacker Complexity Ein Angreifer besitzt limitierte Rechen- und Speicherkapazität.

Strong Cryptography Es werden nur kryptographisch sichere Verfahren ohne bekannte Schwachstellen genutzt. Insbesondere kann nur jemand, der im Besitz des privaten Schlüssels ist, gültige Signaturen erzeugen.

Während der Analyse betrachten wir folgende Bedrohungen:

Man-in-the-Middle Es existiert ein MitM, der sich zwischen den einzelnen Parteien befindet und Nachrichten lesen, manipulieren und senden kann.

Sofern eine Verbindung mit sicherem Kontext zwischen einer Relying Party und einem WebAuthn Client besteht, ist es diesem Angreifer nicht möglich, die in dieser Verbindung gesendeten Nachrichten zu lesen, manipulieren oder zusätzliche zu senden. Ansonsten kann dieser, wie oben beschrieben, agieren.

Nonkonformität Eine der Parteien führt Überprüfungen nicht korrekt durch und verhält sich somit nicht konform.

4.2 Analyse

In diesem Abschnitt zeigen wir die Notwendigkeit der auszuführenden Überprüfungen einer konformen Partei während einer Registrierung bzw. Authentifizierung. Dabei untersuchen wir die Auswirkungen auf die Sicherheitsziele, wenn einzelne Überprüfungen von einer nicht konformen Partei nicht korrekt ausgeführt werden. Zusätzlich betrachten wir, welche Gefahr von einem MitM ausgeht, der sich zwischen den Parteien befindet.

4.2.1 Relying Party

In diesem Kapitel betrachten wir die Folgen einer sich nicht konform verhaltenden Relying Party und untersuchen, welche Sicherheitsziele dadurch nicht mehr gewährleistet werden können. Wir betrachten ebenfalls die Auswirkungen beim Verwenden von verschiedenen Optionen von WebAuthn.

4.2.1.1 Überprüfung der Challenge während einer Registrierung

Angenommen die Relying Party überprüft während einer Registrierung nicht korrekt, ob die erhaltene Challenge der zu erwartenden Challenge entspricht. Dadurch wäre folgendes Szenario möglich. Dabei gehen wir davon aus, dass ein CSRF Angriff möglich ist:

1. Alice startet eine Registrierung bei einer Relying Party.
2. Eve registriert einen Account bei der Relying Party und startet die Registrierung eines weiteren öffentlichen Schlüssels für diesen Account.
3. Eve fängt die Antwort von Alice WebAuthn Client ab und sendet im Rahmen der Registrierung des weiteren öffentlichen Schlüssels Alice Antwort (und damit Alice öffentlichen Schlüssel) an die Relying Party.
4. Die Relying Party registriert Alice öffentlichen Schlüssel im Kontext mit Eves Account.
5. Eve startet einen *CSRF* Angriff, um bei Alice eine Authentifizierung zu starten und um zu Verschleiern, dass Alice Registrierung nicht erfolgreich war.
6. Alice wird aufgefordert die Authentifizierung an ihrem Authenticator zu autorisieren.
7. Alice kann sich erfolgreich einloggen. Dabei ist sie in dem Glauben, sich in ihren neu registrierten Account eingeloggt zu haben und nicht in Eves.
8. Alice und Eve haben Zugriff auf den Account.

Daher kann *Authentication* nicht mehr gewährleistet werden.

Das gleiche Szenario ergibt sich, wenn die Challenge von der Relying Party nicht serverseitig generiert wird.

4.2.1.2 Überprüfung des Origin während einer Registrierung

Angenommen die Relying Party überprüft während einer Registrierung nicht korrekt, ob der erhaltene Origin dem zu erwartenden Origin entspricht. Dadurch wäre folgendes Szenario möglich:

1. Alice möchte sich bei einer Relying Party mit dem Origin “https://example.com” registrieren.
2. Fälschlicherweise besucht sie Eves Origin “https://eve.example.com”.
3. Eve registriert einen Account bei der Relying Party in Alice Namen, startet die Registrierung eines weiteren öffentlichen Schlüssels für diesen Account und übergibt die Optionen an Alice.
4. Als RPID kann Eve ebenfalls “example.com” nutzen.
5. Alice autorisiert die Registrierung und antwortet Eve.
6. Eve leitet die Antwort an die Relying Party weiter.
7. Die Relying Party registriert Alice öffentlichen Schlüssel im Kontext mit Eves Account.

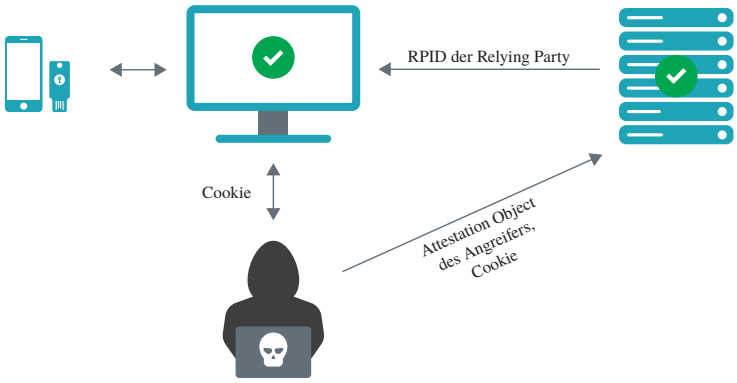


Abbildung 4.1: Kein Token Binding während Registrierung

8. Alice kann sich nun erfolgreich bei der Relying Party authentifizieren. Dabei ist sie in dem Glauben sich in ihren Account einzuloggen und nicht in Eves. Hierfür muss Eve nicht als MitM agieren.
9. Alice und Eve haben Zugriff auf den Account.

Dies ist beispielsweise auch möglich, wenn die Relying Party den Origin “https://example.com:28197” und der Angreifer “https://example.com:2998” nutzt. In diesem Fall muss die Relying Party und der Angreifer als RPID “example.com” verwenden.

Daher kann *Authentication* nicht mehr gewährleistet werden.

4.2.1.3 Überprüfung des Token Binding während einer Registrierung

Angenommen die Relying Party verwendet während einer Registrierung kein Token Binding oder überprüft dieses nicht korrekt. Dadurch wäre folgendes Szenario möglich (siehe Abbildung 4.1):

1. Alice möchte sich bei einem bereits bestehenden Account bei einer Relying Party einen weiteren öffentlichen Schlüssel registrieren.
2. Eve ist im Besitz des Cookies, der bei der Verbindung zwischen Alice WebAuthn Client und der Relying Party genutzt wird.
3. Eve antwortet der Relying Party mit einem eigenen öffentlichen Schlüssel, indem sie den Cookie in einer eigenen Verbindung mitsendet.
4. Die Relying Party registriert Eves öffentlichen Schlüssel im Kontext mit Alice Account.
5. Eve hat Zugriff auf Alice Account.

Daher kann *Authentication* nicht mehr gewährleistet werden.

Diese Bedenken bestehen insbesondere, wenn die Relying Party nicht zwischen den Sitzungen ihrer Nutzer separiert und der Angreifer somit kein Cookie für das oben beschriebene Szenario benötigt.

4.2.1.4 Überprüfung der RPID

Da die Relying Party während einer Registrierung oder Authentifizierung bereits den Origin überprüft, ist die Überprüfung der RPID vermutlich eine Plausibilitätsprüfung: Ein Origin kann mehrere RPIDs zulassen (siehe Kapitel 4.2.3.5).

4.2.1.5 Überprüfung der Anwesenheit einer Person bzw. Verifizierung des Nutzers während einer Registrierung

Angenommen die Relying Party überprüft während einer Registrierung nicht korrekt, ob die Anwesenheit einer Person getestet bzw. die Verifizierung des Nutzers durchgeführt wurde. Dadurch kann eine Registrierungen ohne Zustimmung des Nutzers durchgeführt und somit *User Consent* nicht mehr gewährleistet werden.

4.2.1.6 Überprüfung der Attestation Signature

Angenommen die Relying Party überprüft während einer Registrierung nicht korrekt, ob die erhaltene Attestation Signature eine gültige Signatur ist. Dadurch kann die Relying Party nicht die Eigenschaften und das Vertrauenslevel des verwendeten Authenticator prüfen. Dabei beruht die Vertrauenswürdigkeit der Signatur auf das beim Signieren eingesetzte kryptographische Schema, auf welches wir in der Analyse nicht näher eingehen werden.

Daher kann *Registration Verifiability* nicht mehr gewährleistet werden.

Für eine erfolgreiche Verifizierung muss die Relying Party ebenfalls den öffentlichen Schlüssel der Attestation Signature verifizieren können - beispielsweise durch eine Zertifizierungsstelle.

Diese Signatur schützt **nicht** vor einem MitM Angriff (siehe Kapitel 4.2.2.1 bzw. 4.2.2.2).

4.2.1.7 Überprüfung der Credential ID

Angenommen die Relying Party überprüft während einer Registrierung nicht korrekt, ob die erhaltene Credential ID bereits vergeben ist. Dadurch wäre folgendes Szenario möglich:

1. Eve registriert sich bei einer Relying Party mit Alice Credential ID.
2. Eve startet eine Authentifizierung und übergibt in ihrer Antwort Alice User Handle.
3. Die Relying Party verifiziert die erhaltene Antwort mit Eves öffentlichen Schlüssel und überprüft ob, die erhaltene User Handle zur Credential ID gehört.
4. Die Relying Party authentifiziert Eve als Alice.

Daher kann *Authentication* nicht mehr gewährleistet werden.

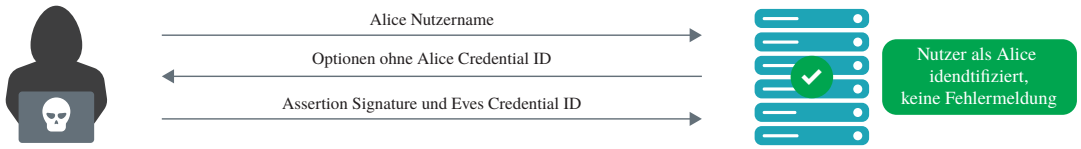


Abbildung 4.2: Fehlender User Handle Angriff

4.2.1.8 Überprüfung von “allowCredentials”

Angenommen die Relying Party überprüft während einer Authentifizierung nicht korrekt, ob die erhaltene Credential ID erlaubt ist. Dadurch wäre folgendes Szenario möglich:

1. Eve stiehlt Alice Authenticator.
2. Alice widerruft den auf dem Authenticator gespeicherten privaten Schlüssel bei der Relying Party.
3. Eve startet eine Authentifizierung in Alice Namen.
4. Die Relying Party übergibt *allowCredentials*, in denen der widerrufene Schlüssel nicht aufgeführt wird.
5. Eve setzt *allowCredentials* zurück und führt mithilfe des gestohlenen Authenticators eine Authentifizierung durch. Dabei testet der Authenticator lediglich die Anwesenheit einer Person.
6. Die Relying Party authentifiziert Eve als Alice.

Daher kann *Authentication* nicht mehr gewährleistet werden.

4.2.1.9 Überprüfung des User Handle

Angenommen die Relying Party überprüft während einer Authentifizierung nicht korrekt, ob der erhaltene User Handle zur erhaltenen Credential ID gehört. Dadurch wäre folgendes Szenario möglich:

1. Eve und Alice sind beide bei einer Relying Party registriert.
2. Eve führt eine Authentifizierung durch, in der sie ihren privaten Schlüssel verwendet. In der Antwort übergibt sie zwar ihre Credential ID, aber Alice User Handle.
3. Die Relying Party verifiziert die erhaltene Signatur mit dem öffentlichen Schlüssel von Eve und authentifiziert Eve anhand des User Handle als Alice.

Daher kann *Authentication* nicht mehr gewährleistet werden. Dieses Szenario wird in Issue #753 [Lun18] erwähnt.

4.2.1.10 Fehlender User Handle Angriff

Die Spezifikation definiert nicht, welcher Nutzer authentifiziert wird, wenn die Relying Party während einer Authentifizierung in der entsprechenden Antwort keinen User Handle erhält. Dadurch ist folgendes Szenario möglich:

1. Eve und Alice sind beide bei einer Relying Party registriert.
2. Eve startet eine Authentifizierung und behauptet Alice zu sein, indem sie Alice Nutzername übergibt.
3. Die Relying Party identifiziert den zu authentifizierenden Nutzer als Alice. In den Optionen verwendet sie nicht *allowCredentials* und übergibt daher nicht Alice registrierte Credential ID.
4. Eve führt die Authentifizierung mit ihrem privaten Schlüssel durch. In ihrer Antwort übergibt sie ihre Credential ID, aber keinen User Handle.
5. Die Relying Party verifiziert die Assertion Signature erfolgreich mit Eves öffentlichem Schlüssel. Da weder *allowCredentials* verwendet wird, noch ein User Handle in der Antwort enthalten ist, authentifiziert die Relying Party den bereits identifizierten Nutzer. Somit kann sich Eve als Alice authentifizieren.

Daher kann *Authentication* nicht mehr gewährleistet werden.

Dieses Problem habe ich in Issue #1078 [Stö18a] erwähnt. Die Autoren der Spezifikation haben dieses Problem anerkannt und haben vor, folgende Änderung vorzunehmen:

Falls der zu authentifizierende Nutzer bereits identifiziert ist, wird von der Relying Party überprüft, ob dieser Nutzer der Besitzer der erhaltenen Credential ID ist. Ansonsten muss ein User Handle übergeben werden.

4.2.1.11 Überprüfung der Challenge während einer Authentifizierung

Angenommen die Relying Party überprüft während einer Authentifizierung nicht korrekt, ob die erhaltene Challenge der zu erwartenden Challenge entspricht. Dadurch wäre folgendes Szenario möglich:

1. Eve ist im Besitz der Antwort von Alice WebAuthn Client aus einer früheren Authentifizierung.
2. Eve startet eine Authentifizierung und sendet die gespeicherte Antwort, die Alice Credential ID und eine Assertion Signature enthält.
3. Die Relying Party verifiziert die erhaltene Signatur mit dem öffentlichen Schlüssel von Alice und authentifiziert Eve als Alice.

Daher kann *Authentication* nicht mehr gewährleistet werden.

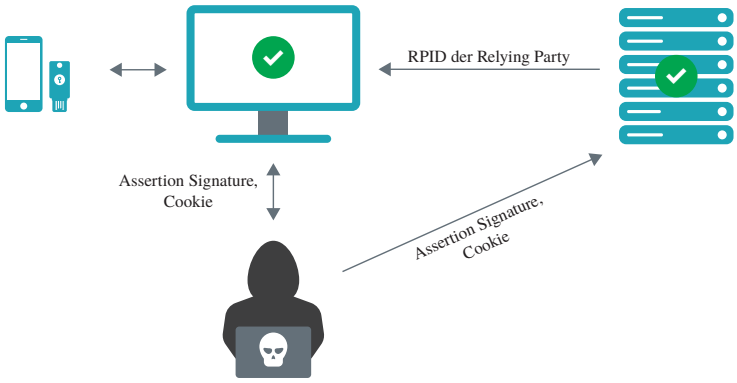


Abbildung 4.3: Kein Token Binding während Authentifizierung

4.2.1.12 Überprüfung des Origin während einer Authentifizierung

Angenommen die Relying Party überprüft während einer Authentifizierung nicht korrekt, ob der erhaltene Origin dem zu erwartenden Origin entspricht. Dadurch wäre der in Kapitel 4.2.2.3 beschriebene MitM Angriff möglich, sofern die Relying Party und der Angreifer die gleiche RPID nutzen.

Dies ist beispielsweise der Fall, wenn die Relying Party den Origin “https://example.com” bzw. “https://example.com:11018” und der Angreifer “https://eve.example.com” bzw. “https://example.com:7301” nutzt. In diesen Fällen kann bzw. muss die Relying Party und der Angreifer als RPID “example.com” verwenden.

Daher kann *Authentication* nicht mehr gewährleistet werden.

4.2.1.13 Überprüfung des Token Binding während einer Authentifizierung

Angenommen die Relying Party verwendet kein Token Binding. Dadurch wäre folgendes Szenario möglich (siehe Abbildung 4.3):

1. Alice möchte sich bei einer Relying Party authentifizieren.
2. Bei der Verbindung zwischen Alice WebAuthn Client und der Relying Party wird ein Cookie eingesetzt.
3. Eve ist im Besitz dieses Cookies und der Antwort von Alice WebAuthn Client.
4. Eve kann die Antwort unter Verwendung des Cookies in einer eigenen Verbindung an die Relying Party senden.
5. Die Relying Party authentifiziert Eve als Alice.

Daher kann *Authentication* nicht mehr gewährleistet werden.

Diese Bedenken bestehen insbesondere, wenn die Relying Party nicht zwischen den Sitzungen ihrer Nutzer separiert und der Angreifer somit kein Cookie für das oben beschriebene Szenario benötigt.

4.2.1.14 Überprüfung der Anwesenheit einer Person während einer Authentifizierung

Angenommen die Relying Party überprüft während einer Authentifizierung nicht korrekt, ob die Anwesenheit einer Person getestet wurde. Dadurch wäre folgendes Szenario möglich:

1. Eve kann über ein Skript auf Alice Authenticator zugreifen und führt eine Authentifizierung durch. Dabei testet der Authenticator nicht die Anwesenheit einer Person.
2. Eve sendet die Antwort des Authenticators an die zugehörige Relying Party.
3. Die Relying Party authentifiziert Eve als Alice.

Daher kann *Authentication* nicht mehr gewährleistet werden.

4.2.1.15 Überprüfung Verifizierung des Nutzers während einer Authentifizierung

Angenommen die Relying Party überprüft während einer Authentifizierung nicht korrekt, ob eine Verifizierung des Nutzers durchgeführt wurde. Dadurch wäre folgendes Szenario möglich:

1. Eve stiehlt Alice Authenticator.
2. Eve führt mithilfe des gestohlenen Authenticators eine Authentifizierung durch. Dabei testet der Authenticator lediglich die Anwesenheit einer Person.
3. Die Relying Party authentifiziert Eve als Alice.

Daher kann *Authentication* nicht mehr gewährleistet werden.

4.2.1.16 Überprüfung der Erweiterungen

Angenommen die Relying Party überprüft während einer Registrierung oder Authentifizierung nicht korrekt, welche Erweiterungen durchgeführt wurden. Dadurch kann ein Angreifer, der die Liste der auszuführenden Erweiterungen ändern kann, zusätzliche Erweiterungen ausführen lassen. Die folgenden Bedenken gelten ebenso, wenn die Relying Party die Erweiterungen veranlasst:

Durch die Erweiterungen werden persönliche Informationen über den Nutzer bekannt, wie beispielsweise die geographische Position oder Eigenschaften des Authenticators. Diese Informationen könnten einen Nutzer als Besitzer mehrerer Accounts bei verschiedenen Relying Parties identifizieren. Die Erweiterungen sind daher eine Abwägung zwischen *Authentication* und *Privacy*.

4.2.1.17 Überprüfung der Assertion Signature

Angenommen die Relying Party überprüft während einer Authentifizierung nicht korrekt, ob die erhaltene Assertion Signature eine gültige Signatur ist. Dadurch könnte ein Angreifer eine gültige Antwort auf eine beliebige Authentifizierung generieren und sich als ein beliebiger Nutzer ausgeben.

Daher kann *Authentication* nicht mehr gewährleistet werden.

4.2.1.18 Überprüfung des Signature Counters

Angenommen die Relying Party überprüft während einer Authentifizierung nicht korrekt, ob der erhaltene Wert des Signature Counters größer ist als der abgespeicherte. Dadurch könnte ein Angreifer einen geklonten Authenticator nutzen, um sich als dessen Besitzer auszugeben (siehe Kapitel 4.2.5.10).

Daher kann *Authentication* nicht mehr gewährleistet werden. Dieses Szenario wird in der Spezifikation erwähnt.

Anzumerken ist, dass ein Angreifer auf dem geklonten Authenticator so lange Authentifizierungen durchführen kann, bis der Signature Counter groß genug ist. Daher ist diese Überprüfung keine wirksame Maßnahme.

4.2.1.19 Fehlende Überprüfung von "requireResidentKey"

Eine Relying Party möchte nicht, dass der private Schlüssel als Credential ID verschlüsselt wird. Sie kann allerdings nicht feststellen, ob der Authenticator diesem Wunsch nachgekommen ist.

Dieses Problem wurde bereits in Issue #1060 [Shi18b] erwähnt und kann behoben werden, indem die Attestation Signature zusätzlich signiert, ob der private Schlüssel als Credential ID verschlüsselt wurde oder nicht.

4.2.1.20 Fehlende Überprüfung von "pubKeyCredParams"

Die Relying Party überprüft nicht, ob das neu generierte Schlüsselpaar den in *pubKeyCredParams* beschriebenen Anforderungen entspricht.

Widersprechen die Eigenschaften des Schlüsselpaares der *Strong Cryptography*, können keine Ziele gewährleistet werden. Dieses Problem wurde bereits in Issue #1061 [Shi18a] erwähnt und kann behoben werden, indem die Relying Party überprüft, ob der erhaltene öffentlichen Schlüssel den Anforderungen entspricht.

4.2.1.21 Option "attestation"

Eine Relying Party kann das unveränderte, ein verifizierbares oder kein Attestation Object verlangen:

direct Die Relying Party erfährt die AAGUID und den öffentlichen Schlüssel der Attestation Signature. Mithilfe dieser Informationen könnte ein Nutzer als Besitzer mehrerer Accounts bei verschiedenen Relying Parties identifiziert werden.

Daher kann *Privacy* nicht mehr gewährleistet werden.

indirect, none Die Relying Party erhält die ursprüngliche Attestation Signature nicht. Die Bedenken hierzu sind in Kapitel 4.2.1.6 aufgeführt.

Daher kann *Registration Verifiability* nicht mehr gewährleistet werden.

4.2.1.22 Option “excludeCredentials” und “allowCredentials”

Angenommen ein Angreifer übergibt während einer Authentifizierung einer Relying Party einen Nutzernamen und die Relying Party antwortet mit einer Liste von erlaubten Credential IDs (*allowCredentials*). Dadurch erfährt der Angreifer zum einen, dass der angegebene Nutzername registriert ist, zum anderen erhält er die Credential IDs des Nutzers. Die erhaltenen Credential IDs könnten eingesetzt werden, um eine bestimmte Person als der Besitzer eines Accounts zu identifizieren (siehe Kapitel 4.2.3.5 und Kapitel 4.2.4.3).

Ein ähnliches Problem tritt auch während einer Registrierung im Hinblick auf *excludeCredentials* auf. Die Spezifikation trifft keine Aussage darüber, ob diese Option ausschließlich bei bereits authentifizierten Nutzern genutzt werden darf. Daher ist es möglich, dass ein Angreifer bei einer Registrierung Credential IDs übergeben bekommt.

4.2.1.23 Multi-Faktor-Authentifizierung

WebAuthn kann sowohl als einzelner Faktor, als auch alleine für eine Multi-Faktor-Authentifizierung eingesetzt werden. Dabei ist der erste Faktor immer der Besitz des Authenticators, der zweite und gegebenenfalls dritte die auf dem Authenticator durchgeführte Verifizierung des Nutzers. Die Verifizierung des Nutzers kann mehrere Faktoren für eine erfolgreiche Verifizierung fordern:

Etwas, das du besitzt Dieser Faktor wird durch den Besitz des Authenticators und dem Signieren eines erfolgreichen Tests der Anwesenheit einer Person bzw. einer erfolgreichen Verifizierung des Nutzers bereitgestellt.

Etwas, das du weißt Dieser Faktor wird durch das Signieren einer erfolgreichen Verifizierung des Nutzers bereitgestellt werden. Dabei wird die Verifizierung des Nutzers beispielsweise durch Abfrage einer PIN durchgeführt.

Etwas, das du bist Dieser Faktor wird durch das Signieren einer erfolgreichen Verifizierung des Nutzers bereitgestellt. Dabei wird die Verifizierung des Nutzers beispielsweise durch einen Fingerabdruckscan durchgeführt.

4.2.2 Kommunikation zwischen Relying Party und WebAuthn Client

In diesem Kapitel betrachten wir, welche Auswirkungen ein MitM hat, der sich zwischen der Relying Party und dem WebAuthn Client befindet, und welche Sicherheitsziele dadurch nicht mehr gewährleistet werden können. Dabei untersuchen wir sowohl den Fall, dass ein sicherer Kontext vorhanden ist, als auch dass keiner vorhanden ist. Letzteres kann eintreten, wenn der WebAuthn Client sich nicht konform verhält oder wenn der Angreifer beispielsweise den privaten Schlüssel des Zertifikats der Relying Party besitzt oder erfolgreich ein Skript im Namen der Relying Party ausführen kann. In dieser Analyse gehen wir auf diesen Punkt nicht näher ein.

Abbildung 4.4: Erfolgloser Phishing Angriff

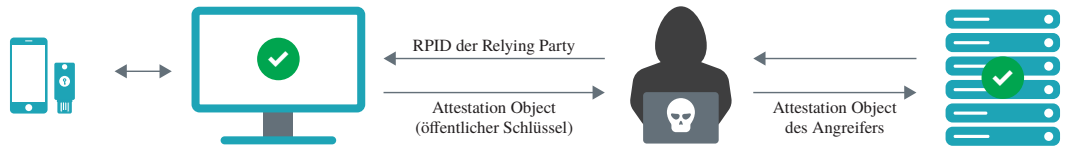


Abbildung 4.5: Man-in-the-Middle Angriff während Registrierung

4.2.2.1 Man-in-the-Middle Angriff mit sicherem Kontext

Angenommen es existiert während einer Registrierung oder Authentifizierung ein MitM, der sich zwischen einer konformen Relying Party und einem konformen WebAuthn Client befindet. Dabei besteht ein sicherer Kontext, d.h. die Verbindung ist über TLS gesichert. Im Folgenden betrachten wir verschiedene Szenarien:

Netzwerkangreifer Ein als Netzwerkangreifer agierender Angreifer kann aufgrund des sicheren Kontexts den Inhalt der Nachrichten nicht lesen oder ändern. Selbst wenn dieser während einer Authentifizierung in den Besitz der Antwort des WebAuthn Clients eines Nutzers gelangt, kann er diese nicht in der selben Authentizierung erfolgreich an die Relying Party senden - sofern die Relying Party zwischen den Sitzungen ihrer Nutzer separiert (siehe Kapitel 4.2.1.13). Ebenso kann er diese aufgrund der Challenge nicht nutzen, um sich in einer selbst initialisierten Authentifizerung als der zugehörige Nutzer zu authentifizieren. Aufgrund von *Strong Cryptography* kann er ebenfalls keine gültige Signatur erzeugen.

Phishing Angriff Angenommen der Angreifer agiert während einer Authentifizierung als eine nicht konforme Relying Party und versucht durch einen Phishing Angriff in den Besitz einer gültigen Assertion Signature für die Authentifizierung bei einer konformen Relying Party zu gelangen (siehe Abbildung 4.4). Der Angreifer kann aufgrund seines Origin und seiner RPID nur Operationen in seinem Namen ausführen. Das bedeutet, dass selbst wenn die Authentifizierung nicht auf dem Authenticator fehlschlägt und der Angreifer eine vom Nutzer ausgestellte Assertion Signature erhält, die Assertion Signature den Origin und RPID des Angreifers signiert und somit die Authentifizierung von einer konformen Relying Party abgebrochen wird.

Übernahme der Sitzung Angenommen der Angreifer kann die Sitzung eines Nutzers übernehmen - beispielsweise da er im Besitz des Sitzungscookies ist, der in der Verbindung eines Nutzers zwischen Relying Party und WebAuthn Client eingesetzt wird. Die Folgen diskutieren wir in Kapitel 4.2.1.3 und 4.2.1.13.

4.2.2.2 Man-in-the-Middle Angriff ohne sicheren Kontext während einer Registrierung

Angenommen es existiert während einer Registrierung ein MitM, der sich zwischen der Relying Party und dem WebAuthn Client befindet und den sicheren Kontext ignorieren kann. Dadurch wäre folgendes Szenario möglich (siehe Abbildung 4.5):

1. Alice möchte sich bei einer Relying Party registrieren.
2. Eve agiert als der erwähnte MitM und ersetzt Alice generiertes Attestation Object mit einem selbst generierten - insbesondere den zu registrierenden öffentlichen Schlüssel.
3. Die Relying Party verifiziert die Antwort und beendet die Registrierung erfolgreich.
4. Eves öffentlicher Schlüssel ist nun zusammen mit Alice Account registriert. Dadurch ergeben sich folgende Szenarien:
   - **A** Alice kann sich nicht mehr in ihren Account einloggen.
   - **B** Eve kann sich beliebig als Alice authentifizieren. Das ist insbesondere ein Problem, wenn Alice keinen neuen Account, sondern bei ihrem bereits bestehenden Account eine zusätzliche Authentifizierungsmöglichkeit registriert hat.
   - **C** Eve agiert auch in Zukunft als MitM. Immer wenn sich Alice bei der Relying Party authentifizieren möchte, ersetzt Eve die von Alice mit dem falschen privaten Schlüssel generierte Antwort mit einer gültigen Antwort. Dabei kann sich Eve beliebig als Alice authentifizieren.

In jedem einzelnen Szenario kann *Authentication* nicht mehr gewährleistet werden.

Die Spezifikation behauptet, dass die oben aufgeführten Szenarien nur auftreten können, wenn entweder *Self Attestation* oder *None Attestation* verwendet wird oder wenn die Relying Party das Attestation Object nicht verifiziert. Ansonsten sei WebAuthn resistent gegenüber diesem Angreifer. Dies ist allerdings **nicht korrekt**.

In Issue #1088 [Stö18b] weise ich auf folgende Problematik hin:

1. Ein Angreifer ist im Besitz eines von der Relying Party akzeptierten sich konform verhaltenden Authenticators.
2. Der Angreifer kann die Anwesenheit einer Person oder die Verifizierung des Nutzers simulieren und somit automatisiert Registrierungen und Authentifizierungen durchführen. Dies kann beispielsweise durch ein Gerät bewerkstelligt werden, welches einen Knopf auf dem Authenticator betätigt oder ein Passwort für die Verifizierung des Nutzers eingibt.
3. Während einer Registrierung bzw. Authentifizierung führt das erwähnte Gerät die Registrierung bzw. Authentifizierung durch.
4. Der Angreifer kann daher das Attestation Object unter Verwendung eines beliebigen Typs unbemerkt austauschen.

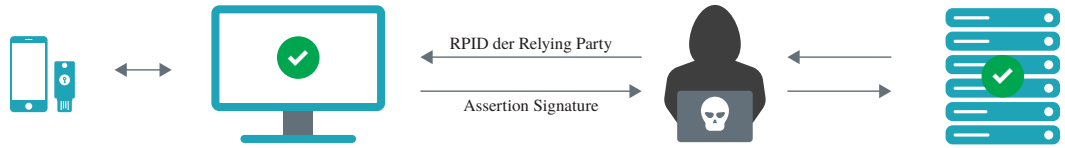


Abbildung 4.6: Man-in-the-Middle während Authentifizierung

Dieses Problem wurde von den Autoren der Spezifikation anerkannt. Zum aktuellen Zeitpunkt überlegen die Autoren aufgrund der genannten Einwände, die oben erwähnte Behauptung bezüglich der Resistenz von MitM Angriffen während einer Registrierung aus der Spezifikation zu entfernen.

Bisher gibt es für dieses Problem noch keine Lösungsvorschläge. Eine Relying Party könnte dem Nutzer die registrierte Credential ID anzeigen oder nur eine AAGUID erlauben, auf die ein Angreifer keinen Zugriff hat. Beide Ansätze sind allerdings nicht praktikabel, da WebAuthn benutzerfreundlich sein und im großen Rahmen von der Öffentlichkeit genutzt werden soll und somit jeder Nutzer Zugriff auf einen akzeptierten Authenticator haben sollte.

4.2.2.3 Man-in-the-Middle Angriff ohne sicheren Kontext während einer Authentifizierung

Angenommen es existiert während einer Authentifizierung ein MitM, der sich zwischen der Relying Party und dem WebAuthn Client befindet und den sicheren Kontext ignorieren kann. Dadurch wäre folgendes Szenario möglich (siehe Abbildung 4.6):

1. Alice möchte sich bei einer Relying Party authentifizieren.
2. Eve agiert als der oben erwähnte MitM, startet eine Authentifizierung bei der Relying Party und übergibt die Optionen an Alice - insbesondere die RPID der Relying Party.
3. Der WebAuthn Client akzeptiert die erhaltene RPID.
4. Alice autorisiert die Authentifizierungsanfrage und antwortet Eve.
5. Eve leitet die erhaltene Antwort an die Relying Party weiter.
6. Die Relying Party authentifiziert Eve als Alice.

Dadurch kann *Authentication* nicht mehr gewährleistet werden.

4.2.3 WebAuthn Client

In diesem Kapitel betrachten wir die Folgen eines sich nicht konform verhaltenden WebAuthn Clients und untersuchen, welche Sicherheitsziele dadurch nicht mehr gewährleistet werden können. Wir betrachten ebenfalls die Auswirkungen beim Verwenden von verschiedenen Optionen von WebAuthn.

Abbildung 4.7: Sicherer Kontext

Abbildung 4.8: Sicherer Kontext und “sameOriginWithAncestors” Überprüfung

4.2.3.1 Überprüfung des sicheren Kontexts

Angenommen der WebAuthn Client stellt während einer Registrierung bzw. Authentifizierung die WebAuthentication API trotz unsicheren Kontext zur Verfügung. Dadurch wäre ein MitM Angriff zwischen der Verbindung zwischen der Relying Party und dem WebAuthn Client möglich (siehe Kapitel 4.2.1.2 und 4.2.2.2 bzw. 4.2.2.3), insbesondere könnte ein Angreifer seinen Origin fälschen (siehe Kapitel 4.2.3.2).

Daher kann *Authentication* nicht mehr gewährleistet werden.

4.2.3.2 Überprüfung des Origins

Angenommen der WebAuthn Client bestimmt während einer Registrierung bzw. Authentifizierung den Origin einer Relying Party nicht korrekt. Dadurch kann ein Angreifer einen beliebigen Origin und damit RPID nutzen und wird somit zu dem in Kapitel 4.2.1.2 und 4.2.2.2 bzw. 4.2.2.3 beschriebenen Angreifer.

Daher kann *Authentication* nicht mehr gewährleistet werden.

Die korrekte Bestimmung verhindert somit den beschriebenen Angriff (siehe Abbildung 4.7).

4.2.3.3 Überprüfung “sameOriginWithAncestors”

Angenommen der WebAuthn Client überprüft während einer Registrierung bzw. Authentifizierung nicht, ob der Origin des aufrufenden Skriptes den gleichen Origin wie alle seine Eltern besitzt. Dadurch kann ein Angreifer den Origin einer Relying Party nutzen, beispielsweise indem er die Website der Relying Party als iframe auf seiner eigenen einbettet. Somit wird er trotz sicherem Kontext zu dem in Kapitel 4.2.1.2 und 4.2.2.2 bzw. 4.2.2.3 beschriebenen Angreifer.

Daher kann *Authentication* nicht mehr gewährleistet werden.

Die korrekte Überprüfung verhindert somit den beschriebenen Angriff (siehe Abbildung 4.8).

4.2.3.4 Überprüfung der RPID

Angenommen der WebAuthn Client überprüft nicht korrekt, ob die erhaltene RPID eine gültige RPID ist. Eine Registrierung oder Authentifizierung würde zwar letztendlich von der Relying Party aufgrund der Überprüfung des Origin abgebrochen, allerdings wäre folgendes Szenario möglich:

1. Alice möchte sich bei Eve authentifizieren.
2. Eve übergibt die RPID einer anderen Relying Party und in *allowCredentials* eine bei der anderen Relying Party registrierte Credential ID.
3. Alice nutzt fälschlicherweise diese Credential ID für die Authentifizierung, da ihr Authenticator keine Informationen anzeigen kann.
4. Eve weiß nun, dass Alice im Besitz der übergebenen Credential ID ist.

Daher kann *Privacy* nicht mehr gewährleistet werden.

4.2.3.5 Geteilte RPID

Relying Parties, die sich gegenseitig nicht vertrauen, können bzw. müssen sich in manchen Fällen die gleiche RPID teilen. Dies ist beispielsweise der Fall, wenn

- eine Relying Party den Origin “https://example.com” und eine weitere “https://eve.example.com” besitzt . Beide Relying Parties können die RPID “example.com” nutzen.
- eine Relying Party den Origin “https://example.com:1337” und eine andere “https://example.com:4242” besitzt. Beide Relying Parties **müssen** die RPID “example.com” nutzen.

Wenn eine Relying Party eine RPID mit einer weiteren Relying Party teilt, kann diese durch *allowCredentials* den Besitzer eines Accounts identifizieren, sofern der Authenticator mit einer Signatur antwortet (siehe Kapitel 4.2.3.4). Dies trifft ebenfalls zu, wenn nur eine Relying Party existiert.

Daher kann *Privacy* nicht gewährleistet werden.

4.2.3.6 Option “excludeCredentials”

Angenommen der WebAuthn Client verzögert nicht den Abbruch der Registrierung in folgenden Fällen:

- Es ist kein Authenticator vorhanden.
- Eine Credential ID aus *excludeCredentials* ist auf dem Authenticator im Kontext der RPID registriert und der Nutzer bricht die Registrierung ab.

Dadurch kann die Relying Party zwischen den beiden Fällen unterscheiden und feststellen, ob der Nutzer im Besitz einer bestimmten Credential ID ist. Somit kann *Privacy* nicht mehr gewährleistet werden. Diese Bedenken werden in der Spezifikation erwähnt.

Da der Nutzer seine Zustimmung gibt, wird die Relying Party benachrichtigt, wenn eine Credential ID aus *excludeCredentials* auf dem Authenticator im Kontext der RPID registriert ist und der Nutzer die Registrierung trotzdem fortführen möchte. Dies wurde eingeführt, damit die Relying Party dem Nutzer anzeigen kann, dass der Authenticator bereits mit dem Account verbunden ist [Cze18].

4.2.3.7 Option "allowCredentials"

Angenommen der WebAuthn Client verzögert nicht den Abbruch der Authentifizierung in folgenden Fällen:

- Der Authenticator bricht die Authentifizierung ab, da keine Credential ID in *allowCredentials* im Kontext der RPID registriert ist.
- Ein Credential ist vorhanden, aber der Nutzer bricht die Authentifizierung auf dem Authenticator ab.

Dadurch kann die Relying Party zwischen den beiden Fällen unterscheiden und feststellen, ob der Nutzer der Besitzer einer bestimmten Credential ID ist. Somit kann *Privacy* nicht mehr gewährleistet werden. Diese Bedenken werden in der Spezifikation erwähnt.

4.2.3.8 Überprüfung der Erweiterungen

Nicht jede Erweiterung ist während einer Registrierung oder Authentifizierung erlaubt. Dies liegt meiner Meinung nach daran, dass manche Erweiterungen während einer Registrierung oder Authentifizierung keinen Mehrwert für die Relying Party haben. Im Folgenden betrachten wir zwei Erweiterungen, deren Einsatz während einer Registrierung problematisch sein könnte.

Mit Hilfe der Erweiterungen *Simple Transaction* und *Generic Transaction* könnte ein MitM Angreifer durch Anzeigen falscher Informationen versuchen, sich als eine andere Relying Party auszugeben. Da der Authenticator allerdings nur Operationen im Kontext mit der RPID des Angreifers durchführt, hat dieser Versuch keine Auswirkungen auf die Sicherheitsziele. Insbesondere werden nach dem Anzeigen der falschen Informationen die richtigen Informationen auf dem Authenticator angezeigt.

4.2.3.9 FIDO AppID Erweiterung

Angenommen der WebAuthn Client überprüft bei der Ausführung der *FIDO AppID* Erweiterung nicht korrekt, ob die erhaltene AppID im Kontext von U2F gültig ist. Dadurch ergeben sich die gleichen Folgen, wie wenn der WebAuthn Client die RPID falsch bestimmt (siehe Kapitel 4.2.3.4).

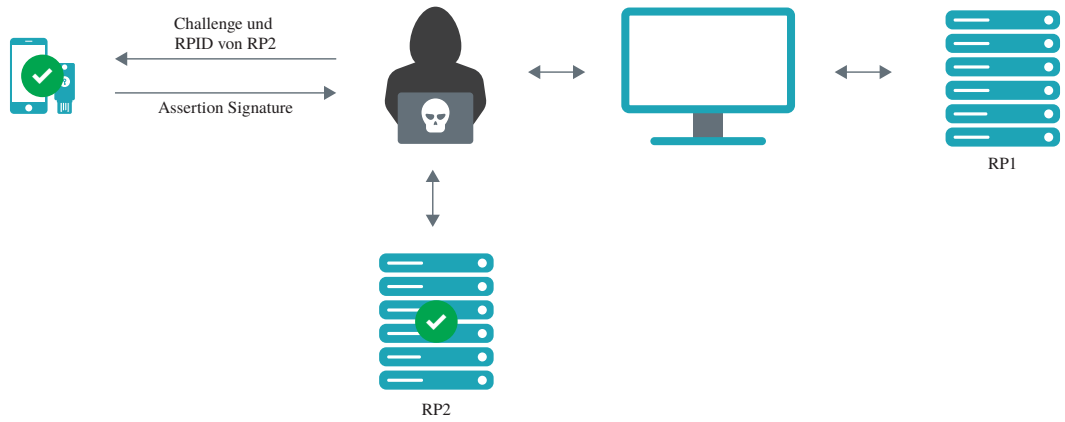


Abbildung 4.9: Man-in-the-Middle Angriff während Registrierung und Authentifizierung

4.2.4 Kommunikation zwischen WebAuthn Client und Authenticator

In diesem Kapitel betrachten wir, welche Auswirkungen ein MitM hat, der sich zwischen dem WebAuthn Client und dem Authenticator befindet, und welche Sicherheitsziele dadurch nicht mehr gewährleistet werden können. Im Wesentlichen unterscheidet sich dieser von dem in Kapitel 4.2.2 darin, dass er nach der Origin Überprüfung des WebAuthn Clients agiert und die Spezifikation - im Gegensatz zur Verbindung zwischen Relying Party und WebAuthn Client - keinen sicheren Kontext vorschreibt.

4.2.4.1 Man-in-the-Middle Angriff während einer Registrierung

Angenommen es existiert während einer Registrierung ein MitM, der sich zwischen dem WebAuthn Client und dem Authenticator befindet und den Inhalt der gesendeten Nachrichten lesen und ändern kann. Dieser Angreifer kann, wie der in Kapitel 4.2.2.2 beschriebene MitM, den öffentlichen Schlüssel eines Nutzers während einer Registrierung austauschen. Zusätzlich wäre folgendes Szenario möglich (siehe Abbildung 4.9):

1. Alice ist bereits bei der Relying Party RP2 registriert und möchte sich bei der Relying Party RP1 registrieren.
2. Eve agiert als der erwähnte MitM und startet währenddessen eine Authentifizierung in Alice Namen bei RP2.
3. Eve ändert entsprechend *clientDataHash* und *rpid* in der Nachricht zwischen Alice WebAuthn Client und Authenticator.
4. Alice Authenticator hat keine Möglichkeit Informationen anzuzeigen. Daher bestätigt Alice fälschlicherweise die Authentifizierung bei RP2.
5. Eve leitet die erhaltene Antwort an RP2 weiter.
6. RP2 authentifiziert Eve als Alice.

Daher können *Authentication* und *User Consent* nicht mehr gewährleistet werden.

4.2.4.2 Man-in-the-Middle Angriff während einer Authentifizierung

Angenommen es existiert während einer Authentifizierung ein MitM, der sich zwischen dem WebAuthn Client und dem Authenticator befindet und den Inhalt der gesendeten Nachrichten lesen und ändern kann. Dadurch wäre folgendes Szenario möglich:

1. Alice möchte sich bei einer Relying Party authentifizieren.
2. Eve agiert als der erwähnte MitM und startet ebenfalls eine Authentifizierung in Alice Namen bei der Relying Party.
3. Eve ändert entsprechend *clientDataHash* und *rpid* in der Nachricht zwischen Alice WebAuthn Client und Authenticator.
4. Alice kann zwischen ihrer und Eves Authentifizierungsanfrage nicht differenzieren und autorisiert die Authentifizierung.
5. Eve leitet die abgefangene Antwort an die Relying Party weiter.
6. Die Relying Party authentifiziert Eve als Alice.

Daher können *Authentication* und *User Consent* nicht mehr gewährleistet werden.

4.2.4.3 Option “excludeCredentials” und “allowCredentials”

Angenommen es existiert ein MitM während einer Registrierung, der sich zwischen dem WebAuthn Client und dem Authenticator befindet und den Inhalt der gesendeten Nachrichten lesen und ändern kann. Dieser kann im Gegensatz zur Relying Party zwischen den verschiedenen Fällen bei *excludeCredentials* und *allowCredentials* unterscheiden und somit feststellen, ob ein Nutzer der Besitzer einer bestimmten Credential ID ist. Dies liegt daran, dass er nicht von der Verschleierung des WebAuthn Client betroffen ist (siehe Kapitel 4.2.3.6 und 4.2.3.7).

Insbesondere kann er die Listen *excludeCredentials* und *allowCredentials* beliebig ändern.

Daher kann *Privacy* nicht mehr gewährleistet werden.

4.2.5 Authenticator

In diesem Kapitel betrachten wir die Folgen eines sich nicht konform verhaltenden Authenticators und untersuchen, welche Sicherheitsziele dadurch nicht mehr gewährleistet werden können. Wir betrachten ebenfalls die Auswirkungen beim Verwenden von verschiedenen Optionen von WebAuthn.

4.2.5.1 Option “excludeCredentials”

Angenommen der Authenticator führt eine Registrierung durch, obwohl eine Credential ID aus *excludeCredentials* im Kontext der gegebenen RPID auf dem Authenticator registriert ist. Dies kompromittiert das Konzept für jede Credential ID eines Accounts einen anderen Authenticator zu haben. Dies ist beispielsweise bei dem Verlust eines Authenticators von Vorteil.

4.2.5.2 Überprüfung der RPID

Angenommen der Authenticator ignoriert, dass ein Schlüssel nur im Kontext der abgespeicherten RPID genutzt werden darf. Dadurch ergibt sich das in Kapitel 4.2.3.4 beschriebene Szenario: ein Angreifer könnte feststellen, ob ein Nutzer der Besitzer einer bestimmten Credential ID ist.

Daher kann *Privacy* nicht mehr gewährleistet werden.

Authentication ist nicht betroffen, da der Origin der aufrufenden Relying Party signiert wird.

4.2.5.3 Bedenken Attestation Key

Ein Attestation Key kann Rückschlüsse auf den Nutzer enthalten. Dies ist beispielsweise bei *Basic Attestation* der Fall.

Wenn ein Authenticator einen einzigartigen Attestation Key von dem Typ *Basic Attestation* und somit einen einzigartigen zugehörigen öffentlichen Schlüssel besitzt, kann der Nutzer leicht über mehrere Accounts hinweg bei einer Registrierung mit diesem Authenticator identifiziert werden. Daher schlägt FIDO in *Authenticator Security Requirements* [Lin17] vor, dass mindestens 100 000 Authenticatoren den selben Attestation Key nutzen. Dadurch wäre der zugehörige öffentliche Schlüssel nicht mehr einzigartig.

Um zu verhindern, dass mehrere Authenticatoren einen gemeinsamen Attestation Key nutzen, könnte *Attestation CA* eingesetzt werden. Doch auch hier kann der Aussteller des Zertifikats Rückschlüsse auf den Nutzer enthalten, wenn der Aussteller beispielsweise nur für einen stark eingeschränkten Kreis von Nutzern zur Verfügung steht.

Eine weitere Option wäre das Verwenden von *ECDAA* und somit das Verwenden von anonymen Signaturen. Laut Paragon [Par18] sollte diese allerdings erst genutzt werden, wenn bestehende technische Probleme mit *ECDAA* behoben sind.

Daher kann unter Umständen *Privacy* nicht mehr gewährleistet werden. Diese Bedenken sind in der Spezifikation erwähnt.

4.2.5.4 Authenticator ohne Anzeige

Angenommen der Authenticator hat nicht die Möglichkeit Informationen während einer Registrierung oder Authentifizierung anzuzeigen. Dadurch kann sich der Nutzer dem Kontext nicht mehr bewusst sein. Ihm ist es nicht möglich zwischen einer Registrierung oder Authentifizierung zu unterscheiden oder auf wessen Veranlassung diese durchgeführt werden sollen.

Daher kann *User Consent* nicht mehr gewährleistet werden.

4.2.5.5 Nutzerzustimmung

Angenommen der Authenticator setzt die Markierung der Anwesenheit einer Person bzw. Verifizierung des Nutzers, obwohl eine entsprechende Überprüfung nicht durchgeführt wurde. Dadurch könnte ein Angreifer beispielsweise durch ein Skript auf einem Authenticator Registrierung bzw. Authentifizierung im Namens des Besitzers durchführen. Die Folgen diskutieren wird in den Kapiteln 4.2.1.5, 4.2.1.14 und 4.2.1.15.

Daher kann *User Consent* und somit *Authentication* nicht mehr gewährleistet werden.

4.2.5.6 Nicht signierte Credential ID und User Handle

In der Antwort einer Authentifizierung ist die verwendete Credential ID sowie der User Handle nicht signiert enthalten. Da die Relying Party bei einer veränderten Credential ID einen falschen öffentlichen Schlüssel für die Verifizierung verwendet und die Authentifizierung damit abbricht, ist dies kein Problem. Bei einem veränderten User Handle bricht die Authentifzierung ebenfalls ab, da der User Handle im Kontext der Credential ID gespeichert sein muss.

Diese Bedenken sind in der Spezifikation erwähnt.

4.2.5.7 Data Protection

Angenommen der Authenticator schützt die auf ihm gespeicherten Informationen nicht ausreichend genug. Dadurch wäre es einem Angreifer möglich, den privaten Schlüssel eines Nutzers auszulesen und sich mit diesem als der Nutzer auszugeben.

Daher kann *Authentication* nicht mehr gewährleistet werden.

4.2.5.8 Anwesenheit einer Person

Angenommen der Authenticator besitzt nur die Möglichkeit die Anwesenheit einer Person zu testen. Dadurch wäre es einem Angreifer mit physikalischen Zugriff auf den Authenticator möglich, eine Authentifizierung im Namen des Besitzers durchzuführen.

Daher kann *Authentication* nicht mehr gewährleistet werden.

4.2.5.9 Verifizierung des Nutzers

Angenommen der Authenticator besitzt nur die Möglichkeit eine Verifizierung des Nutzers ohne biometrische Verifizierung durchzuführen - beispielsweise durch Eingabe einer PIN. Dadurch wäre es einem Angreifer mit physikalischen Zugriff auf den Authenticator möglich, unter Kenntnis der PIN eine Authentifizierung im Namen des Besitzers durchzuführen. Die PIN könnte ein Angreifer durch einen Phishing Angriff in Erfahrung bringen.

Daher kann *Authentication* nicht mehr gewährleistet werden.

4.2.5.10 Geklonter Authenticator

Angenommen ein Angreifer ist im Besitz eines geklonten Authenticators. Je nach Eigenschaft des Authenticators kann der Angreifer sich als der Besitzer ausgeben (siehe Kapitel 4.2.1.18, 4.2.5.8 und 4.2.5.9). Ein geklonter Authenticator ist ein Authenticator, von dem ein Angreifer erfolgreich ein Abbild erstellen konnte.

Daher kann *Authentication* nicht mehr gewährleistet werden.

4.2.5.11 Signature Counter

Angenommen der Authenticator unterstützt nur einen *globalen* Signature Counter. Wenn der zugehörige Wert einzigartig unter allen Authenticatoren ist, kann ein Nutzer als Besitzer verschiedener Accounts bei verschiedenen Relying Parties identifiziert werden.

Daher kann *Privacy* nicht mehr gewährleistet werden.

4.2.5.12 Eigenschaft “platform” und “cross-platform”

Das Verwenden eines *platform* oder *cross-platform* Authenticators kann Einfluss auf die Sicherheit von WebAuthn nehmen.

platform Diese Art von Authenticatoren hat den Vorteil, dass sie eine interne Kommunikation nutzen und kritische Nachrichten nicht über beispielsweise eine drahtlose Verbindung gesendet werden. Allerdings kann es vorkommen, dass der WebAuthn Client beispielsweise einen Fingerabdruckscanner für den Authenticator bereitstellt und daher an der Verifizierung des Nutzers beteiligt ist. Somit erfährt der Informationen über den Nutzer. Des Weiteren könnte ein solcher Authenticator von mehreren Nutzern genutzt werden. Dabei sollten nur die berechtigten Nutzer auf einen bestimmten privaten Schlüssel zugreifen können.

Die letzten zwei Bedenken sind in der Spezifikation erwähnt.

cross-platform Ein Nachteil dieser Art von Authenticatoren ist die genutzte externe Kommunikation. Vorteilhaft ist allerdings die nur auf dem Authenticator ausgeführte Verifizierung des Nutzers und der eingeschränkte Zugriff auf das Gerät selber, welches wiederum aufgrund der Portabilität leichter gestohlen werden kann.

4.3 Ergebnis

Im Folgenden diskutieren wir basierend auf der vorausgegangenen Analyse, welche Sicherheitsziele aus Kapitel 4.1 von WebAuthn gewährleistet werden können und welche nicht. Wir nehmen an, dass das in Kapitel 4.2.1.10 erwähnte Problem bereits mit dem Lösungsvorschlag der Autoren der Spezifikation behoben ist.

Authentication Um sich als ein Nutzer bei einer konformen Relying Party zu authentifizieren, muss die Relying Party eine Assertion Signature erfolgreich mit dem öffentlichen Schlüssel des Nutzers verifizieren. Der öffentliche Schlüssel wird über die erhaltene Credential ID identifiziert. Dabei wird überprüft, ob diese im Kontext mit dem zu authentifizierenden Nutzer gespeichert ist. Die Verifizierung ist nur erfolgreich, wenn die Assertion Signature eine gültige Signatur des Origin und RPID der Relying Party sowie der für diese Authentifizierung generierte Challenge ist. Sofern der WebAuthn Client des Nutzers sich konform verhält und ein sicherer Kontext besteht, kann solch eine Signatur aufgrund des Origin nur auf Veranlassung der Relying Party angefragt werden. Lediglich derjenige Authenticator, der das asymmetrische Schlüsselpaar während der Registrierung generiert hat, hat Zugriff auf den zugehörigen privaten Schlüssel des Nutzers und kann somit diese Signatur generieren. Hierfür nehmen wir an, dass der Authenticator des Nutzers sich konform verhält, dass einem Angreifer die für eine Autorisierung benötigten Informationen nicht bekannt sind und dass *User Consent* gegeben ist. Somit ist der private Schlüssel des Nutzers vor unautorisierten Zugriffen geschützt wird.

Durch das Signieren der Challenge kann keine Assertion Signature mehrfach verwendet werden. Aufgrund unserer Annahme *Strong Cryptography* und des sicheren Kontexts kann kein MitM in den Besitz einer gültigen Assertion Signature gelangen, indem er diese beispielsweise anfrägt, abfängt oder erzeugt. Hierfür nehmen wir an, dass die Verbindung zwischen WebAuthn Client und Authenticator gegen einen MitM geschützt ist.

Zusätzlich kann durch das Signieren des Token Binding vor einem MitM geschützt werden, der im Besitz der Antwort des WebAuthn Client eines Nutzers ist und dessen Sitzung übernimmt. Hierfür nehmen wir an, dass die Relying Party zwischen den Sitzungen ihrer Nutzer separiert.

Somit kann sich nur die Person als der Nutzer authentifizieren, die im Besitz des Authenticators ist und auf diesem Operationen autorisieren kann.

Falls es während der Registrierung keinen erfolgreichen Angriff gegeben hat, wird das Sicherheitsziel *Authentication* gewährleistet.

Privacy Um einen Nutzer als Besitzer eines weiteren Accounts bei einer oder der gleichen Relying Party zu identifizieren, müssen die während einer Registrierung bzw. Authentifizierung ausgetauschten Informationen der beiden Accounts in irgendeinem Bezug zueinander stehen.

Falls Informationen einzigartig für den Authenticator des Nutzers ist und es somit keinen anderen Nutzer gibt, der diese während einer Registrierung bzw. Authentifizierung zurückgibt, kann der Nutzer leicht als Besitzer eines weiteren Accounts identifiziert werden. Zu diesen Informationen gehören:

- Attestation Type bzw. der öffentliche Schlüssel des Attestation Key
- AAGUID
- *globaler* Signature Counter
- Ausgaben der Erweiterungen (beispielsweise geographische Position)

Alle anderen Informationen werden für jeden Account jeweils aufgrund von *Strong Cryptography* neu mithilfe von kryptographisch sicheren Funktionen von dem konformen Authenticator des Nutzers generiert.

Aufgrund der Überprüfung der RPID und Verschleierung von Fehlermeldungen durch einen konformen WebAuthn Client, wird die Gefahr einer Identifizierung durch *excludeCredentials* und *allowCredentials* behoben. Dabei betrachtet ein konformer Authenticator nur private Schlüssel, die im Kontext mit der RPID abgespeichert sind. Es besteht allerdings kein Schutz wenn *User Consent* nicht gegeben ist. In diesem Fall kann eine Relying Party, die sich eine RPID mit einer weiteren Relying Party teilt, oder ein MitM, der sich zwischen dem WebAuthn Client und dem Authenticator befindet, einen Besitzer identifizieren. Insbesondere erfährt die Relying Party, dass ein privater Schlüssel im Kontext der RPID auf dem Authenticator registriert ist, wenn der Nutzer die Registrierung fortführt und nicht abbricht, obwohl die zugehörige Credential ID in *excludeCredentials* übergeben wird.

Daher kann *Privacy* nur gewährleistet werden, wenn der WebAuthn Client und Authenticator der Nutzers sich konform verhalten, *User Consent* gegeben ist, die Verbindung zwischen diesen beiden Parteien gegen ein MitM geschützt ist, keine Erweiterungen durchgeführt werden, ein Signature Counter *pro Schlüsselpaar* eingesetzt und ein Attestation Typ verwendet wird, der keine Rückschlüsse auf den Besitzer zulässt.

User Consent Eine erfolgreiche Registrierung oder Authentifizierung kann nur durchgeführt werden, wenn die jeweilige Operation auf dem Authenticator durch die Anwesenheit einer Person oder die Verifizierung des Nutzers autorisiert und dies in der jeweiligen Signatur bestätigt wird. Hierfür nehmen wir an, dass der Authenticator des Nutzers sich konform verhält.

Ein Nutzer kann sich beim Autorisieren einer Operation dem Kontext nur bewusst sein, wenn der Authenticator die RPID und, im Falle einer Authentifizierung, den zu authentifizierenden Account anzeigt.

In diesem Fall wird *User Consent* gewährleistet.

Ansonsten können beim Authenticator eingehende Nachrichten durch einen MitM oder nicht konformen WebAuthn Client unbemerkt geändert werden. Somit ist der Nutzer sich dem Kontext nicht bewusst und *User Consent* kann nicht gewährleistet werden. Ist *User Consent* nicht gewährleistet, kann auch *Authentication* nicht mehr gewährleistet werden!

Registration Verifiability Eine Relying Party verifiziert die Integrität der während einer Registrierung erhaltenen Informationen anhand der erhaltenen Attestation Signature. Die Verifizierung ist nur erfolgreich, wenn die Attestation Signature eine gültige Signatur des Origin und der RPID der Relying Party sowie der für diese Registrierung generierte Challenge ist. Die Konformität des genutzten Authenticator wird von der Relying Party anhand der erhaltenen AAGUID evaluiert.

Die Authentizität und somit *Registration Verifiability* kann nicht gewährleistet werden, da die Attestation Signature nur Aussagen über den verwendeten Authenticator trifft. Die Relying Party kann nicht feststellen, ob dabei der Authenticator des Nutzers oder eines Angreifers verwendet wurde.

Die Spezifikation behauptet, dass *Registration Verifiability* bei einem MitM gewährleistet wird, der den sicheren Kontext ignorieren kann oder sich zwischen dem WebAuthn Client und Authenticator befindet - sofern die Attestation Signature nicht den Typ *Self Attestation* oder *None Attestation* besitzt. **Dies trifft nicht zu**. Solch ein Angreifer kann den zu registrierenden öffentlichen Schlüssel unbemerkt austauschen, da dieser ebenfalls Zugriff auf einen konformen Authenticator hat und somit eine gültige Attestation Signature eines beliebigen Typs generieren kann (siehe Kapitel 4.2.2.2).

Dieses Problem wurde von den Autoren der Spezifikation anerkannt. Zum aktuellen Zeitpunkt überlegen die Autoren aufgrund der genannten Einwände, die oben erwähnte Behauptung bezüglich der Resistenz von MitM Angriffen während einer Registrierung aus der Spezifikation zu nehmen.

5 Fazit

Die Verwendung eines Passwortes für die Authentifizierung ist insbesondere für Phishing Angriffe anfällig. Kommt ein Angreifer in den Besitz eines Passwortes, kann er sich beliebig als der Nutzer ausgeben. Eine Lösung für dieses Problem ist der Einsatz mehrerer Faktoren: Der Nutzer authentifiziert sich beispielsweise über etwas, das er weiß, und über etwas, das er besitzt.

WebAuthentication (WebAuthn) ist ein Entwurf eines Standards, der auf eine benutzerfreundliche Weise einem Nutzer ermöglicht, sich bei einer Relying Party unter Einsatz von mehreren Faktoren passwortlos zu registrieren und authentifizieren. Die Benutzerfreundlichkeit ist ein wichtiger Punkt. Wenn die Verwendung des Authentifizierungsverfahren zu kompliziert für einen Nutzer ist, wird sich das Verfahren nicht durchsetzen können.

In dieser Arbeit untersuchen wir die Sicherheit dieses Protokolls. Hierfür beschreiben wir zunächst den Ablauf einer Registrierung und Authentifizierung und implementieren eine Relying Party. Anschließend führen wir eine informelle Sicherheitsanalyse durch.

Eine Relying Party besitzt eine RPID, über die sie identifiziert wird. Durch die Separierung der RPID von dem Origin der Relying Party, kann eine Relying Party mehrere Origins nutzen. Dabei leitet sich die RPID von den Origins der Relying Party ab.

Während einer Registrierung generiert eine Relying Party eine Nonce, die sogenannte Challenge, und einen Identifikator für den Nutzer, die sogenannte User Handle, und übergibt diese dem WebAuthn Client des Nutzers. Die Verbindung zwischen der Relying Party und dem WebAuthn Client besitzt einen sicheren Kontext, d.h. sie ist über HTTPS gesichert. Der WebAuthn Client stellt sicher, dass eine Relying Party keine Operationen im Namen einer anderen Relying Party durchführen kann. Die Challenge wird vom WebAuthn Client zusammen mit dem Origin an den Authenticator des Nutzers gesendet. Auf diesem muss der Nutzer die Registrierung autorisieren. Hierfür testet der Authenticator die Anwesenheit einer Person oder führt eine Verifizierung des Nutzers durch. Anschließend wird ein asymmetrisches Schlüsselpaar generiert. Der zugehörige private Schlüssel wird auf dem Authenticator zusammen mit dem User Handle gespeichert. Der zugehörige öffentliche Schlüssel wird als Teil einer Antwort zusammen mit einer Attestation Signature an die Relying Party gesendet. Diese verifiziert die Antwort und speichert die erhaltenen Informationen im Kontext des zu registrierenden Nutzers. Die Attestation Signature signiert den öffentlichen Schlüssel, einen Identifikator des Schüsselpaares, den Origin, die RPID sowie einen Identifikator des Authenticator-Modells. Eine Attestation Signature besitzt einen Typ, der bestimmt, welche Art von Schlüssel für das Signieren verwendet wurde. Hierzu kann beispielsweise der neu generierte private Schlüssel verwendet werden. Des Weiteren besitzt sie ein Format, das den Kontext angibt, in dem die Signatur generiert wurde. In dieser Arbeit abstrahieren wir von dem Konzept des Formats.

Während einer Authentifizierung generiert die Relying Party eine Challenge und sendet diese über den WebAuthn Client an den Authenticator des Nutzers. Auf diesem muss der Nutzer die Authentifizierung, wie während der Registrierung, autorisieren. Der gespeicherte private Schlüssel signiert

die Challenge, den Origin und die RPID. Diese sogenannte Assertion Signature wird zusammen mit dem User Handle an die Relying Party gesendet. Sofern die Signatur die zu erwartenden Werte signiert und mit dem registrierten öffentlichen Schlüssel des Nutzers verifiziert werden kann, wird der Nutzer über den User Handle identifiziert und gegenüber der Relying Party authentifiziert.

Die Relying Party kann zusätzlich Erweiterungen durchführen. In diesen kann sie unter anderem die geographische Position des Authenticators anfragen. Dabei muss die Relying Party damit rechnen, dass die Erweiterungen ignoriert und nicht ausgeführt werden.

Um die Relying Party "Ascensus" zu implementieren, setzen wir einen Node.js Server auf. Ascensus ermöglicht ihren Nutzern, sich im Rahmen von WebAuthn mithilfe eines Authenticators zu registrieren und authentifizieren, der das U2F Protokoll verwendet. Die Verbindung zum Server ist über ein selbst signiertes TLS Zertifikat gesichert. Der Server besitzt eine lokale nicht persistente MongoDB Datenbank und einen CSRF Schutz. MongoDB wird automatisch beim erstmaligen Starten des Servers heruntergeladen und installiert. Jeder Besucher der Internetseite erhält einen signierten Sitzungscookie, der als Identifikator für die in der Datenbank gespeicherten Sitzungsdaten dient und nur über HTTPS verwendet werden kann.

Als WebAuthn Client verwenden wir den Browser Firefox Quantum 61.0.1 zusammen mit Microsoft Windows 10, als Authenticator ein im Browser enthaltenen Softtoken. Nach der Entwicklung wurde die Demo erfolgreich mit einem Yubikey 4 anstelle des Softtoken getestet.

Um sich zu registrieren, besucht ein Nutzer die Startseite "https//ascensus.com". Auf dieser sendet er seinen Nutzernamen über ein Formular an den Server. Wir müssen auf das Verwenden eines Nutzernamens zurückgreifen, da der zu verwendende Authenticator keinen User Handle speichern kann. Serverseitig wird eine 32 byte lange Challenge generiert und mit zusätzlichen Optionen an den Nutzer gesendet. Der Nutzer wird daraufhin aufgefordert, die Registrierung durch Betätigen eines Knopfes auf dem Authenticator zu autorisieren. Im Hintergrund wird von der Relying Party ein Skript ausgeführt. Das Skript ist für den Aufruf der WebAuthentication API mit den generierten Optionen, für das Parsen der Nachrichten und senden der Antwort des WebAuthn Clients an den Server verantwortlich. Dabei stellt der eingesetzte Browser die WebAuthentication API bereit und übernimmt die Kommunikation mit dem Authenticator. Serverseitig kann die vom Skript erhaltene Antwort verifiziert und der erhaltene öffentliche Schlüssel im Kontext des Nutzers in der Datenbank gespeichert werden. Für die Verifizierung wird unter anderem die Challenge aus den Sitzungsdaten geladen. Des Weiteren müssen Teile der erhaltenen Antwort auf Byte-Ebene decodiert werden. Ascensus verlangt keine Attestation Signature.

Im Falle einer Authentifizierung übergibt der Server nicht nur eine Challenge, sondern auch den Identifikator des Schlüsselpaares des Nutzers. Hierfür wird der Nutzer in der Datenbank über seinen Nutzernamen gesucht. Als Antwort erhält die Relying Party eine Assertion Signature. Falls diese mit dem öffentlichen Schlüssel des Nutzers verifiziert werden kann und die zu erwartenden Werte signiert, ist der Nutzer authentifiziert. Der authentifizierte Nutzer wird persistent in der Session gespeichert.

Ein bereits authentifizierter Nutzer kann einen weiteren öffentlichen Schlüssel bzw. einen weiteren Authenticator registrieren. Um sicherzustellen, dass verschiedene Authenticatoren verwendet werden, wird während der Registrierung zusätzlich der Identifikator des bereits registrierten Schlüsselpaares übergeben. Der zusätzliche öffentliche Schlüssel wird zwar in unserer Implementierung verifiziert, allerdings nicht in der Datenbank gespeichert.

In einer informellen Sicherheitsanalyse untersuchen wir die Sicherheit des Protokolls, indem wir aufführen, wieso die während einer Registrierung bzw. Authentifizierung durchzuführenden Überprüfungen der einzelnen Parteien notwendig sind. Des Weiteren untersuchen wir die Kommunikationskanäle im Hinblick auf einen MitM und welche Gefahr von diesem ausgeht. Dabei untersuchen wir das Protokoll auf unter den folgenden Annahmen auf Sicherheitsziele, die wir weiter unten diskutieren: Wir nehmen an, dass ein Angreifer limitierte Rechen- und Speicherkapazität besitzt und dass nur kryptographisch sichere Verfahren eingesetzt werden. Dadurch kann insbesondere kein Angreifer gültige Signaturen ohne den zugehörigen privaten Schlüssel erzeugen.

Während der Analyse verweisen wir auf einen möglichen Angriff auf das Authentifizierungsverfahren: Wenn die Relying Party den zu authentifizierenden Nutzer vor der Authentifizierung bereits identifiziert hat und während der Authentifizierung keinen User Handle erhält, kann ein Angreifer sich als der identifizierte Nutzer authentifizieren.

Ein Ablauf des Angriffes könnte wie folgt aussehen: Um sich als ein Nutzer zu authentifizieren, übergibt der Angreifer während der Authentifizierung beispielsweise den Nutzernamen des Nutzers. Die Relying Party identifiziert dadurch den zu authentifizierenden Nutzer. Der Angreifer signiert die Challenge mit seinem privaten Schlüssel und übergibt in der Antwort an die Relying Party keinen User Handle. Die Relying Party verifiziert die Assertion Signature erfolgreich mit dem öffentlichen Schlüssel des Angreifers. Da weder *allowCredentials* verwendet wird, noch ein User Handle in der Antwort enthalten ist, authentifiziert die Relying Party den bereits identifizierten Nutzer. Somit kann sich der Angreifer als der Nutzer authentifizieren.

Dieses Problem wurde von den Autoren der Spezifikation anerkannt. Aufgrund dessen haben die Autoren der Spezifikation vor, folgende Änderung an dem Protokoll vorzunehmen: Falls der zu authentifizierende Nutzer bereits identifiziert ist, wird von der Relying Party überprüft, ob dieser Nutzer der Besitzer des öffentlichen Schlüssels ist. Ansonsten muss ein User Handle übergeben werden.

Des Weiteren widerlegen wir die in der Spezifikation aufgeführte Behauptung, dass WebAuthn während einer Registrierung resistent gegenüber einem MitM sei, der den sicheren Kontext ignorieren kann. Die Spezifikation begründet dies dadurch, dass ein Angreifer keine gültige Attestation Signature erzeugen kann, die nicht vom Typ *Self Attestation* oder *None Attestation* sind. Dies ist nicht korrekt. Ein Angreifer kann zwar keine Signaturen fälschen, aber er kann sie mit einem eigenen Authenticator automatisiert erzeugen lassen, indem er die Anwesenheit einer Person oder die Verifizierung des Nutzers simuliert. Dies kann durch ein Gerät bewerkstelligt werden, welches einen Knopf auf dem Authenticator betätigt oder auf diesem eine PIN eingibt. Somit ist es ihm möglich während einer Registrierung den öffentlichen Schlüssel eines Nutzers auszutauschen.

Dieses Problem wurde von den Autoren der Spezifikation anerkannt. Zum aktuellen Zeitpunkt überlegen die Autoren aufgrund der genannten Einwände, die oben erwähnte Behauptung bezüglich der Resistenz gegenüber MitM Angriffen während einer Registrierung aus der Spezifikation zu nehmen.

Wie angekündigt, diskutieren wir nun die Sicherheitsziele. Das erste Sicherheitsziel *Authentication* besagt, dass ein Angreifer sich nicht in den Account eines anderen Nutzers einloggen kann. Wir gehen davon aus, dass unser oben erwähnter Angriff durch die Lösung der Autoren der Spezifikation verhindert wird. Dieses Sicherheitsziel wird dadurch gewährleistet, dass nur die Person unter gegebenen *User Consent* (siehe unten) eine Authentifizierung auf dem Authenticator ausführen kann, die diesen während der Registrierung verwendet hat. Hierfür nehmen wir an, dass der

WebAuthn Client und Authenticator des Nutzers sowie die Relying Party sich konform verhalten, dass die Verbindung zwischen WebAuthn Client und Authenticator sicher ist und dass während der Registrierung kein erfolgreicher Angriff stattgefunden hat. Durch das Signieren von Origin, RPID, Challenge und Token Binding während einer Authentifizierung schützt sich die Relying Party erfolgreich gegen einen MitM, der einen Phishing Angriff durchführt oder eine Signatur aus einer älteren Authentifizierung sendet. Hierfür nehmen wir an, dass die Verbindung zwischen der Relying Party und dem WebAuthn Client einen sicheren Kontext besitzt und dass die Relying Party zwischen den Sitzungen ihrer Nutzer separiert.

Das Sicherheitsziel *Privacy* besagt, dass ein Nutzer nur so viele persönliche Informationen preisgibt wie nötig. Insbesondere kann ein Nutzer nicht als der Besitzer eines weiteren Accounts identifiziert werden. Wenn der Authenticator während einer Registrierung oder Authentifizierung einzigartige Informationen bereitstellt, ist dies allerdings möglich. Zu diesen Informationen gehört der öffentliche Schlüssel des Attestation Keys, die AAGUID, ein *globaler* Signature Counter und die Ausgaben der Erweiterungen (beispielsweise geographische Position). Des Weiteren kann durch die Optionen *allowCredentials* und *excludeCredentials* eine Relying Party den Besitzer eines Nutzers feststellen, wenn dieser fälschlicherweise die jeweilige Operation autorisiert und einen der übergebenen Identifikatoren nutzt. Dies ist insbesondere der Fall, wenn eine geteilte RPID existiert oder das nächste Sicherheitsziel *User Consent* nicht gegeben ist. Angenommen es existieren zwei Relying Parties, die sich in ihrem Origin nur im Port unterscheiden, dann müssen sich diese die gleiche RPID teilen. Daher kann das Sicherheitsziel *Privacy* nicht immer gewährleistet.

User Consent besagt, dass ein Nutzer sich dem Kontext einer Operation bewusst sein muss. Das bedeutet, dass dieser unter Kenntnis der RPID weiß, ob durch seine Zustimmung eine Registrierung oder Authentifizierung autorisiert wird und, im Falle einer Authentifizierung, für welchen Account er sich authentifiziert. Da Authenticatoren ohne Anzeige in WebAuthn akzeptiert werden, kann dieses Sicherheitsziel nicht immer gewährleistet werden. Ein Nutzer kann ohne Anzeige nicht wissen, für was er seine Zustimmung gibt. Unter Annahme eines konformen WebAuthn Client und einer sicheren Verbindung zu dem Authenticator könnte der WebAuthn Client das Anzeigen des Kontextes übernehmen.

Schließlich diskutieren wir das letzte Sicherheitsziel *Registration Verifiability*. Dieses besagt, dass die Relying Party die Integrität und Authentizität der erhaltenen Informationen während einer Registrierung verifizieren kann. In der Analyse gehen wir auf die Tatsache ein, dass die Attestation Signature lediglich Aussagen über den verwendeten Authenticator trifft - jedoch nicht darüber, ob dieser Authenticator auch tatsächlich dem Nutzer gehört. Daher kann *Registration Verifiability* nicht gewährleistet werden.

Die in der Einführung erwähnten Probleme eines Passwortes im Hinblick auf Phishing Angriffen und Wiederverwenbarkeit können durch WebAuthn erfolgreich gelöst werden. WebAuthn bietet ein sicheres Authentifizierungsverfahren, welches resistent gegenüber Phishing Angriffen und Wiederverwenbarkeit ist. Insbesondere ist es möglich, das Protokoll aufgrund der Verifizierung des Nutzers auf dem Authenticator und dem gespeicherten User Handle für eine Multi-Faktor-Authentifizierung zu verwenden, ohne dass ein Nutzer einen Nutzernamen oder Passwort der Relying Party übergeben muss. Des Weiteren bietet das Protokoll abhängig von dem eingesetzten Authenticator Privatsphäre in dem Hinblick, dass ein Nutzer nicht als Besitzer mehrerer Accounts identifiziert werden kann. Dies wird dadurch erreicht, dass für jeden Account ein neues asymmetrisches Schlüsselpaar generiert wird.

Um *Authentication* zu gewährleisten, treffen wir in unserer Diskussion mehrere Annahmen. Als nächster Schritt sollte untersucht werden, inwieweit diese erfüllt werden können. Dabei ist insbesondere die Sicherheit der Verbindung zwischen WebAuthn Client und Authenticator – beispielsweise über USB oder NFC – und somit *User Consent* von Bedeutung. Des Weiteren sollte das Protokoll im Hinblick auf die in dieser Arbeit abstrahierten Formate einer Attestation Signature untersucht werden und es sollte ein formeller Beweis für *Authentication* geführt werden.

Literaturverzeichnis

[BBL17] D. Balfanz, A. Birgisson, J. Lang. *FIDO U2F JavaScript API*. 11. Apr. 2017. URL: https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-u2f-javascript-api-v1.2-ps-20170411.html (zitiert auf S. 41).

[BCH18] D. Balfanz, A. Czeskis, J. Hodges. *Web Authentication: An API for accessing Public Key Credentials Level 1. W3C Candidate Recommodation*. W3C. 20. März 2018. URL: https://www.w3.org/TR/2018/CR-webauthn-20180807/ (zitiert auf S. 17, 19, 39).

[Cze18] A. Czeskis. *RPs cannot show "You've Already Registered This Authenticator"Message*. 16. Feb. 2018. URL: https://github.com/w3c/webauthn/issues/806 (zitiert auf S. 63).

[GH18] I. B. Guirat, H. Halpin. „Formal Verification of the W3C Web Authentication Protocol". In: *Proceedings of the 5th Annual Symposium and Bootcamp on Hot Topics in the Science of Security*. HoTSoS '18. Raleigh, North Carolina: ACM, 2018, 6:1–6:10. ISBN: 978-1-4503-6455-3. DOI: 10.1145/3190619.3190640. URL: http://doi.acm.org/10.1145/3190619.3190640 (zitiert auf S. 16, 29).

[JK18] C. Jacomme, S. Kremer. „An Extensive Formal Analysis of Multi-factor Authentication Protocols". In: *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. Juli 2018, S. 1–15. DOI: 10.1109/CSF.2018.00008 (zitiert auf S. 17).

[LCD18] R. Lindemann, J. Cmenisch, M. Drijvers. *FIDO ECDAA Algorithm*. FIDO Alliance. 27. Feb. 2018. URL: https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-ecdaa-algorithm-v2.0-id-20180227.html (zitiert auf S. 29).

[LHBB18] R. Lindemann, B. Hill, D. Balfanz, D. Baghdasaryan. *FIDO AppID and Facet Specification*. 27. Feb. 2018. URL: https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-appid-and-facets-v2.0-id-20180227.html (zitiert auf S. 35).

[Lin17] R. Lindemann. *FIDO Authenticator Security Requirements*. 14. Juni 2017. URL: https://fidoalliance.org/specs/fido-security-requirements-v1.0-fd-20170524/fido-authenticator-security-requirements_20170524.html (zitiert auf S. 66).

[Lin18] R. Lindermann. *FIDO Security Reference*. FIDO Alliance. 27. Feb. 2018. URL: https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-security-ref-v2.0-id-20180227.html (zitiert auf S. 17, 47).

[Lun18] E. Lundberg. *RP assertion algorithm does not say to validate the credential ID*. 22. Jan. 2018. URL: https://github.com/w3c/webauthn/issues/753 (zitiert auf S. 52).

[Mas17] Mastercard. *Mastercard Biometric Card*. 2017. URL: https://www.mastercard.us/en-us/merchants/safety-security/biometric-card.html (zitiert auf S. 20).

[Mer18a] Mercedes. *Connected Vehicle API*. 17. Jan. 2018. URL: https://developer.mercedes-benz.com/apis/connected_vehicle_experimental_api/ (zitiert auf S. 20).

[Mer18b] Mercedes. *Connected Vehicle API Authentication*. 17. Jan. 2018. URL: https://developer.mercedes-benz.com/apis/connected_vehicle_experimental_api/docs#_authentication (zitiert auf S. 20).

[NMC17] E. Navara, L. McCormick, A. Carey. *Web authentication and Windows Hello*. Microsoft. 2. Aug. 2017. URL: https://docs.microsoft.com/en-us/microsoft-edge/dev-guide/device/web-authentication (zitiert auf S. 22, 23).

[Par18] Paragon Initiative Enterprises. *Security Concerns Surrounding WebAuthn: Don't Implement ECDAA (Yet)*. 23. Aug. 2018. URL: https://paragonie.com/blog/2018/08/security-concerns-surrounding-webauthn-don-t-implement-ecdaa-yet (zitiert auf S. 16, 66).

[PNB18] A. Popov, M. Nystroem, D. Balfanz. *The Token Binding Protocol Version 1.0. draft-ietf-tokbind-protocol-19*. 23. Mai 2018. URL: https://tools.ietf.org/html/draft-ietf-tokbind-protocol-19#token-binding (zitiert auf S. 12, 26).

[PP18] A. Powers, Potch. *Web Authentication API*. Mozilla. 28. Juni 2018. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Authentication_API (zitiert auf S. 22).

[SB17] S. Srinivas, D. Balfnaz. *Universal 2nd Factor (U2F) Overview*. FIDO Alliance. 11. Apr. 2017. URL: https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-u2f-javascript-api-v1.2-ps-20170411.html (zitiert auf S. 13).

[Shi18a] K.-E. Shin. *Missing steps of checking pubKeyCredParams during registration step at RP server*. 10. Sep. 2018. URL: https://github.com/w3c/webauthn/issues/1061 (zitiert auf S. 56).

[Shi18b] K.-E. Shin. *No way to verify requireResidentKey during registration step at RP sideNo way to verify requireResidentKey during registration step at RP side*. 10. Sep. 2018. URL: https://github.com/w3c/webauthn/issues/1060 (zitiert auf S. 56).

[Stö18a] M. Stötzner. *Clarify which user to authenticate if userHandle is not present Clarify which user to authenticate if userHandle is not present*. 20. Sep. 2018. URL: https://github.com/w3c/webauthn/issues/1078 (zitiert auf S. 53).

[Stö18b] M. Stötzner. *Leap of Faith not only for Self and None Attestation Types*. 28. Sep. 2018. URL: https://github.com/w3c/webauthn/issues/1088 (zitiert auf S. 59).

[Stö18c] M. Stötzner. *WebAuthn-Node.js-Demo*. 2018. URL: https://github.com/milesstoetzner/WebAuthn-Node.js-Demo (zitiert auf S. 37).

[W3C18] W3C. *Worldline Demo of Web Payments and Web Authentication*. Juni 2018. URL: https://www.w3.org/2018/06/worldline.html (zitiert auf S. 20).

[Wes18] M. West. *Credential Management Level 1*. W3C. 19. Juni 2018. URL: https://w3c.github.io/webappsec-credential-management/#same-origin-with-its-ancestors (zitiert auf S. 26).

[WHA18] WHATWG. *HTML Living Standard*. 19. Sep. 2018. URL: https://html.spec.whatwg.org/multipage/origin.html#concept-origin (zitiert auf S. 12).

Alle URLs wurden zuletzt am 11. 11. 2018 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift