Neue Serie: Continuous Architecture S. 37

Neue Serie: Modell und Sprache in DDD S. 42

Neue Serie: Transaktionskosten S. 60

Java™ magazin

Java | Architektur | Software-Innovation

AGILE

Ein Sehnsuchtsort

entwickler.de

Ausgabe 12.2024

Passkeys und ihre Herausforderungen in der Anwendung

Zukunft ohne Passwörter?

Passwörter könnten bald Geschichte sein – dank Passkeys. Wir wollen nicht nur einen schnellen Überblick über deren Funktionsweise geben, sondern auch in die Tiefe blicken. Wir zeigen, was du als Entwickler wissen solltest, wenn du deine Anwendung oder Website für eine passwortlose Zukunft rüsten willst.

von Christoph Eifert und Jens Le

In einer digitalen Welt, in der Sicherheitsbedrohungen immer herausfordernder werden, steht auch der Mensch im Fokus von Bedrohungsbetrachtungen. Unsichere Praktiken wie das manuelle Eingeben von Passwörtern eröffnen Angreifern potenzielle Angriffspunkte, um unbefugt in fremde Systeme einzudringen. Im Gegensatz dazu bieten Passkeys eine moderne und sichere Alternative, die das Risiko solcher Angriffe minimiert.

Obwohl Passkeys relativ neu erscheinen, da sie erst in den letzten ein bis zwei Jahren eine größere Verbreitung finden, basieren sie im Wesentlichen auf dem bereits 2018 eingeführten FIDO2-Standard. Er besteht hauptsächlich aus zwei Komponenten: WebAuthn (siehe Kasten „WebAuthn“) und CTAP (siehe Kasten „CTAP“). Aufbauend auf diesen Standards versprechen Passkeys, langfristig traditionelle Passwörter durch eine sicherere und benutzerfreundlichere Methode zu ersetzen, um die Identität eines Benutzers zu verifizieren. Doch wie funktionieren Passkeys genau, und warum sollten Unternehmen und Nutzer diese Technologie in Betracht ziehen?

Im Wesentlichen kann der Ablauf der Benutzerauthentifizierung mit Hilfe von Passkeys in zwei Schritten beschrieben werden: der Registrierung und dem Log-in. Beide sind auf Einfachheit und Sicherheit ausgelegt, um den Übergang zu einer passwortlosen Authentifizierung so reibungslos wie möglich zu gestalten.

So läuft die Registrierung ab ...

Die folgende Beschreibung der Registrierung ist auch in **Abbildung 1** veranschaulicht. Zunächst startet der Benutzer den Registrierungsprozess, indem er beispielsweise auf den REGISTRIEREN-Button klickt. Der Client stellt

dann eine Anfrage an den Server für eine sogenannte Challenge, eine zufällig generierte Zeichenkette, und erhält diese als Antwort vom Server zurück.

Im nächsten Schritt ruft der Client das WebAuthn API auf, um Credentials für die Registrierung zu generieren. Daraufhin wird auf dem Authentikator ein Schlüsselpaar mit einem privaten und einem öffentlichen Schlüssel erstellt. Der private Schlüssel wird zum Signieren der Challenge verwendet und entweder lokal sicher auf dem Gerät des Benutzers gespeichert (z. B. TPM, Secure Enclave) oder bei einem Cloud-Anbieter abgelegt. Der zugehörige öffentliche Schlüssel, eine Credential-ID und die signierte Challenge werden vom API zurückgegeben. In diesem Schritt kann der Benutzer aufgefordert werden, sich durch biometrische Daten oder eine PIN zu verifizieren.

Zum Abschluss der Registrierung sendet der Client das Tripel, bestehend aus öffentlichem Schlüssel, Credential-ID und signierter Challenge, an den Server zurück. Der neu registrierte Benutzer wird von nun an mit der Credential ID und dem öffentlichen Schlüssel für Log-ins verknüpft.

… und so das Log-in

Der Log-in-Flow funktioniert analog zum Registrierungs-Flow (**Abb. 2**): Wenn sich der Benutzer anmelden möchte, startet er den Prozess, indem er auf einen Log-in-Button klickt. Daraufhin sendet der Client eine Anfrage für eine zufällige Challenge an den Server. Der Server generiert eine neue Challenge, die an den Client zurückgeschickt wird.

Der Client ruft nun das WebAuthn API auf, um die passenden Credentials vom Authentikator zu erhalten. Auf dem Authentikator wird der Benutzer dann aufgefordert, sich zu verifizieren. Das kann z. B. über eine biometrische Eingabe mit Fingerabdruckscan geschehen. Mit dem Passkey bzw. dem privaten Schlüssel signiert der Authentikator die Challenge und schickt diese zusammen mit dem Benutzernamen und der Credential ID an den Client zurück.

Um den Log-in-Vorgang abzuschließen, sendet der Client die Informationen, die er vom Authentikator be-

Was ist WebAuthn?

WebAuthn (Web Authentication) ist ein Standard, der es ermöglicht, passwortlose Authentifizierung in Webanwendungen durchzuführen. Es basiert auf der Public-Key-Kryptografie, bei der ein Schlüsselpaar (privater und öffentlicher Schlüssel) erstellt wird. Der private Schlüssel wird sicher auf dem Gerät des Benutzers gespeichert, während der öffentliche Schlüssel auf dem Server hinterlegt wird. Beim Log-in wird eine Challenge vom Server an den Benutzer gesendet, die mit dem privaten Schlüssel signiert und zurückgeschickt wird, um die Identität zu bestätigen [1].

Was ist CTAP?

CTAP (Client to Authenticator Protocol) ist ein Protokoll, das festlegt, wie ein Authentifizierungsgerät (z. B. ein Hardware-Token oder einSmartphone, im FIDO2-Kontext auch als Authentikator bezeichnet) mit einem Client (z. B. einem Browser) kommunizieren soll, um zu einer erfolgreichen Authentifizierung zu gelangen [2].

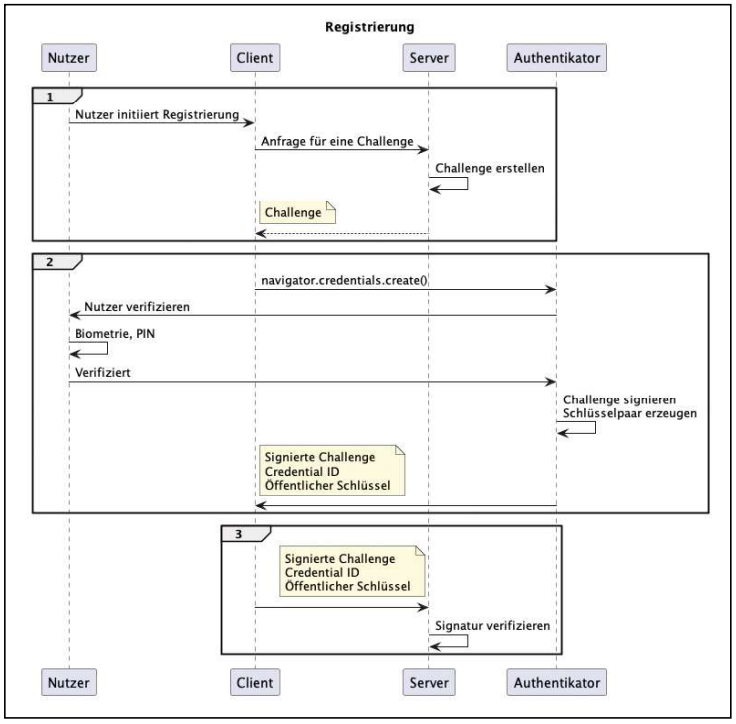

Abb. 1: Ablauf des Registrierungsvorgangs

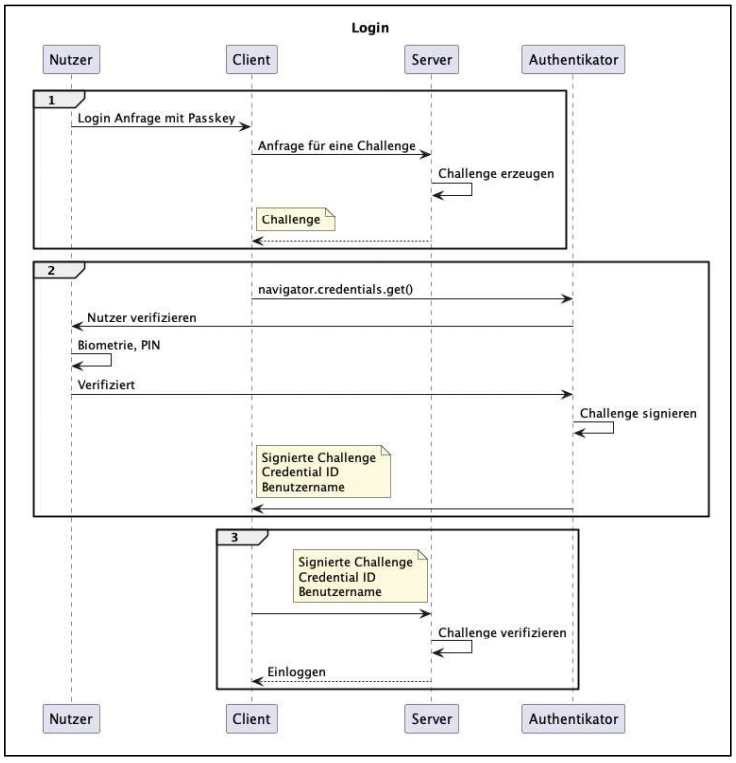

Abb. 2: Ablauf des Log-in-Vorgangs

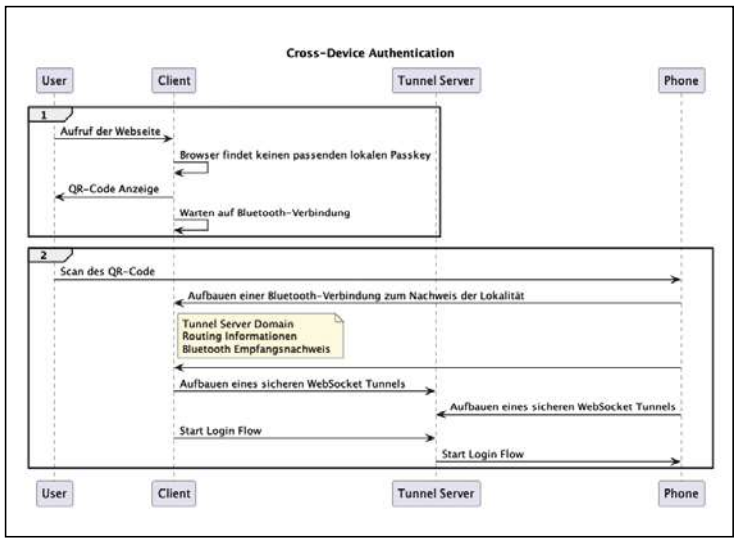

Abb. 3: Ablauf der Cross-Device-Authentifizierung

kommen hat (Benutzername, Credential ID, signierte Challenge) an den Server zurück. Sind dem Server Benutzername und Credential ID bekannt und wurde die Gültigkeit der Challenge mit dem öffentlichen Schlüssel erfolgreich überprüft, wird dem Benutzer der Zugang gewährt.

Passkeys lokal oder in der Cloud speichern

Wichtig für den Umgang mit Passkeys ist die Entscheidung, welche Art der Schlüsselverwaltung gewählt wird. Damit wird auch festgelegt, wo die Passkeys abgelegt werden. Grundsätzlich gibt es zwei Möglichkeiten: Zum einen speichert man sie lokal in einem sicheren Bereich, physikalisch isoliert, mit einem Hardwaremodul (z. B. TPM, Trusted Platform Module, oder USB-Sicherheitstoken). Andererseits ist es auch möglich, sie softwarebasiert in der Cloud abzulegen. Dies bietet den Vorteil einer höheren Flexibilität, da man seine Passkeys auf allen Geräten synchronisieren kann. Es eröffnet aber auch ein neues Einfallstor für Angreifer und kann somit die Sicherheit reduzieren. Im Folgenden werden die Vor- und Nachteile kurz zusammengefasst.

Hardwarebasierte Lösungen bieten eine höhere Sicherheit. Der private Schlüssel verlässt nie das Gerät, was das Phishing-Risiko deutlich verringert. Zudem sind solche Lösungen so konzipiert, dass ein direktes Auslesen oder Kopieren der privaten Schlüssel auf dem Gerät nicht möglich ist. Auch Datenschutzbedenken, die mit der Speicherung sensibler Daten in der Cloud verbunden sind, müssen die Nutzer weniger fürchten, und der Anwender wird unabhängig von Drittanbietern. Allerdings geht dieser Ansatz auch mit einer eingeschränkten Portabilität einher: Wenn ein Nutzer ein neues Gerät verwendet, muss ein neuer Schlüssel generiert und registriert werden, was die Nutzung etwas komplexer machen kann. Entscheidet sich der Benutzer zudem, ausschließlich Passkeys zu verwenden und keine Back-up-Optionen zu nutzen, so geht er das Risiko ein, bei Verlust des Authentikators den Zugang zu verlieren und damit aus der Anwendung ausgesperrt zu werden. Hier empfiehlt es sich, einen zweiten Authentikator als Fallback einzurichten.

Die zweite Möglichkeit zur Speicherung des privaten Schlüssels ist die Nutzung der integrierten Schlüsselverwaltung der Betriebssysteme von Apple (iCloud-Schlüsselbund), Google (Passwortmanager) und Microsoft (Windows Hello), die in der Cloud gespeichert und verwaltet werden. Der Vorteil dieser Lösung ist, dass die Schlüssel über mehrere Geräte hinweg synchronisiert werden können. Das erleichtert den Zugriff auf gesicherte Konten von mehreren Geräten aus und bietet eine nahtlose Benutzererfahrung. Verliert der Nutzer sein Gerät oder wird es beschädigt, kann er sich einfach von einem neuen Gerät aus anmelden, da der private Schlüssel in der Cloud verfügbar ist.

Dadurch wird die Wiederherstellung des Zugangs zu Konten erheblich vereinfacht, es eröffnet aber auch eine weitere Angriffsfläche. Die Speicherung sensibler Daten in der Cloud wirft potenzielle Sicherheitsfragen auf, insbesondere im Hinblick auf den Zugriff durch Dritte oder staatliche Stellen. Obwohl die Schlüssel in der Regel verschlüsselt sind, bleibt die Cloud ein attraktives Ziel für Angreifer.

Wer sich einmal für einen Anbieter entschieden hat, ist an dieses Ökosystem gebunden. Eine Migration, beispielsweise von Windows Hello zu iCloud-Schlüsselbund, ist von den Herstellern nicht vorgesehen. Der Nutzer muss dann die Schlüssel aufwendig neu generieren und speichern lassen.

Komfortabel anmelden: Cross-Device-Authentifizierung

Die Cross-Device-Authentifizierung ist ein wichtiger Baustein für die Praxistauglichkeit von Passkeys. Eine komfortable Anmeldung auf fremden Geräten ist nur damit möglich. In der CTAP2-Spezifikation [2] wird dies als „Hybrid Transport“ bezeichnet (ehemals cloud-assisted Bluetooth Low Energy, kurz caBLE). Wie das im Detail funktioniert, ist für die meisten Anwendungen nicht relevant. Die Umsetzung obliegt hier den Client- (Browser-) bzw. Plattform-(Betriebssystem-) und Authentikator-Anbietern.

Für die Nutzung müssen einige Voraussetzungen erfüllt sein. Beide Geräte müssen Bluetooth 4.0 unterstützen und mit dem Internet verbunden sein. Das externe Gerät muss über eine Kamera zum Scannen des QR-Codes verfügen. In **Abbildung 3** ist der Ablauf des Verbindungsaufbaus dargestellt.

Nach dem Scannen des QR-Codes sendet das externe Gerät eine sogenannte „Bluetooth Low Energy Advertisement“-Nachricht. Sie dient in erster Linie dem Nachweis der Lokalität, d. h., dass sich Client und externes Gerät in unmittelbarer Nähe befinden. Die Advertisement-Nachricht enthält aber auch Informationen für den weiteren Verbindungsaufbau. Mit den Daten aus dem QR-Code und dem Bluetooth Advertisement verfügen sowohl Client als auch externes Gerät über alle

Informationen, um eine sichere Tunnelverbindung zueinander aufzubauen.

Diese Verbindung läuft über einen Proxy-Server, der in der Spezifikation als Tunnel-Service bezeichnet wird. Der Client beginnt die Verbindung mit dem Nachweis, dass er der Empfänger des Bluetooth Advertisements ist, indem er einen Teil der Bluetooth-Nachricht sendet. Nach dem Aufbau der Verbindung beginnt der eigentliche Datenaustausch, z. B. im Fall einer Authentifikation das Senden und Empfangen der Challenge bzw. der signierten Challenge, analog zum WebAuthn-Ablauf bei lokalen Authentikatoren.

Die Implementierung und Bereitstellung des Tunnel-Services übernimmt der bzw. die Plattform des Authentikators. Implementierungen gibt es derzeit nur für iOS und Android (siehe Kasten „Limitierungen der Cross-Device-Authentifizierung"). Da es z. B. bei iOS nicht möglich ist, eigene Servicedaten in „Bluetooth Low Energy Advertisement"-Nachrichten zu packen, muss auch hier die Implementierung der Plattform verwendet werden, wenn man eine eigene Applikation als Cross-Device-Authentikator zur Verfügung stellen möchte. Sowohl iOS als auch Android bieten APIs für die Integration von Drittanbietern, sowohl als Plattform wie auch als Cross-Device.

Registrierung: Deep Dive

Nun haben wir ein grobes Verständnis, wie Registrierung und Log-in in der Theorie funktionieren. Dennoch birgt die konkrete Implementierung eine Reihe von Herausforderungen und Entscheidungen, die Entwickler bzw. Architekten treffen müssen. Daher wollen wir nun die Implementierung genauer unter die Lupe nehmen.

Ausgangspunkt der Registrierung ist der Client, der den Server (im Folgenden entsprechend der Spezifikation Relying Party genannt) anfragt, den Prozess zu starten. Das Web Authentication API macht keine Vorgaben zum Format und dem Inhalt der Anfrage. Typischerweise werden aber die Registrierungsinformationen des Benutzers, z. B. Name und E-Mail-Adresse, mitgesendet. Die Relying Party antwortet mit einem *CredentialCreationOptions*-Objekt. Die Spezifikation dieses Objekts stammt aus dem Credential Management API [3], das unter dem *Navigator*-Objekt des HTML-Standards aufgehängt ist [4]. Die WebAuthn-Spezifikation erweitert das *CredentialCreationOptions*-Objekt um das Feld *publicKey* vom Typ *PublicKeyCredentialCreationOptions* (Listing 1).

Der Server sendet hier Informationen über sich selbst (*PublicKeyCredentialRpEntity*), den Benutzer, für den ein Passkey registriert werden soll (*PublicKeyCredentialUserEntity*), die Challenge, die mit dem generierten Key Pair signiert werden soll und die Information, welche Public-Key-Algorithmen der Server unterstützt und bevorzugt. Zusätzlich gibt es noch optionale Parameter, die die Sicherheit und Benutzerfreundlichkeit beeinflussen können, sowie mögliche Erweiterungen.

Beispielsweise besteht die Möglichkeit, die erlaubten Typen von Authentikatoren und Credential-Arten einzuschränken. Die beiden Arten sind Discoverable (resident) und Non-Discoverable (non-resident) Credentials, deren Unterschied vor allem in der Benutzerfreundlichkeit besteht. *Discoverable* Credentials bieten den Vorteil, dass sich Benutzer direkt authentifizieren können, ohne einen Benutzernamen eingeben zu müssen. Die Verwendung dieser Credentials kann jedoch die Implementierung erschweren und potenziell die Sicherheit verringern, wenn sie nicht richtig gehandhabt werden. In diesem Fall werden die Credentials direkt auf dem Authentikator gespeichert, einschließlich der Metadaten. Besonders wichtig ist hier die Relying Party ID. Sie ermöglicht es dem Client, ohne Benutzerinteraktion zu prüfen, ob Credentials für eine Relying Party verfügbar sind, und wenn ja, dem Benutzer die Authentifizierung mit einem Klick anzubieten.

Die Einschränkung, welche Art von Credential und Authentikator verwendet werden kann, wird über das Objekt *AuthenticatorSelectionCriteria* als Teil von *CredentialCreationOptions* umgesetzt. Beispielsweise kann der Pseudocode in Listing 2 verwendet werden, um sicherzustellen, dass nur Authentikatoren verwendet werden können, die intern (*platform*) zum aktuellen Gerät gehören, eine Benutzerverifikation durchführen und *Discoverable* Credentials zur Verfügung stellen können.

Hierbei ist zu beachten, dass *requireResidentKey* bereits deprecated ist und aus Gründen der Abwärtskompatibilität nur gesetzt werden sollte, wenn *residentKey* als *required* spezifiziert ist. Dieser Code teilt der WebAuthn-API- Implementierung des Clients mit,

Listing 1: PublicKeyCredentialCreationOptions

```
dictionary PublicKeyCredentialCreationOptions      {
  required PublicKeyCredentialRpEntity             rp;
  required PublicKeyCredentialUserEntity           user;
  required BufferSource                            challenge;
  required sequence<PublicKeyCredentialParameters> pubKeyCredParams;
  unsigned long                                    timeout;
  sequence<PublicKeyCredentialDescriptor>          excludeCredentials = [];
  AuthenticatorSelectionCriteria                   authenticatorSelection;
  sequence<DOMString>                              hints = [];
  DOMString                                        attestation = "none";
  sequence<DOMString>                              attestationFormats = [];
  AuthenticationExtensionsClientInputs             extensions;
}
```

Limitierungen der Cross-Device-Authentifizierung

Cross-Device-Authentifizierung wird derzeit nur von Android und iOS unterstützt. Das bedeutet, dass Passkeys auf einem Notebook nicht für das Log-in auf einem Drittgerät verwendet werden können. Auch Ubuntu Mobile bietet derzeit keine Möglichkeit. In der Praxis ist die Notwendigkeit einer Bluetooth-Verbindung ebenfalls einschränkend, da z. B. in einer virtuellen Maschine die Bluetooth-Verbindung des Hosts extra konfiguriert werden muss, was in der Regel nicht standardmäßig der Fall ist.

dass nur Authentikatoren, die alle Anforderungen erfüllen, adressiert werden sollen. *AuthenticatorAttachment* erlaubt es der Relying Party, einzuschränken, dass nur interne (*platform*) oder nur externe (*cross-platform*) Authentikatoren verwendet werden dürfen. Existiert kein Authentikator, der alle Anforderungen erfüllt, wird entweder direkt *cross-device* angeboten oder, falls als *authenticatorAttachment* nur *platform* angegeben ist, eine Fehlermeldung ausgegeben.

Da im nächsten Schritt der Authentikator ausgewählt wird, wäre es spätestens jetzt sinnvoll, den Benutzer darüber zu informieren, welche Authentikatoren tatsächlich zur Verfügung stehen, und eventuell Hinweise zu geben, welcher Authentikator wofür geeignet ist. Auch eine Erklärung, warum ein bestimmter Authentikator nicht verwendet werden kann, ist denkbar, z. B. weil die Relying Party die Eigenschaft *platform* voraussetzt. Um die Benutzerfreundlichkeit zu erhöhen, wäre es wahrscheinlich auch hilfreich, den Benutzer bereits vor Beginn des Registrierungsprozesses darauf hinzuweisen.

Derzeit ist die statische Methode *PublicKeyCredential.isUserVerifyingPlatformAuthenticatorAvailable()* die einzige Möglichkeit für den Client, Informationen über mögliche Authentikatoren zu erhalten. Wie der Name schon sagt, erfahren wir hier nur, ob ein integrierter Plattformauthentikator mit Benutzerverifikation (Pin, Fingerabdruck etc.) vorhanden ist. Das sind z. B. Windows Hello, Apple Touch ID oder Google Password Manager unter Android. Leider ist das derzeit die einzige Information, die Client und Relying Party über die Verfügbarkeit von Authentikatoren erhalten. Das bedeutet, man kann dem Benutzer nicht direkt mitteilen, welche Möglichkeiten er hat. Zum Beispiel, dass kein Authentikator im System verfügbar ist, das System aber Hybridtransport (Cross-Device-Authentifizierung) unterstützt. Der Nutzer erfährt also vom Client nichts und weiß erst beim Start des Log-in-Prozesses, ob und womit er sich anmelden kann. Usability geht anders (siehe Kasten „Privatsphäre und Fingerprinting“).

Im aktuellen „Editor’s Draft“ der WebAuthn-Spezifikation [5] gibt es zusätzlich die Funktion *PublicKeyCredentials.getClientCapabilities()*. Damit würde sich eine Liste der vom Clientsystem unterstützten Funktionen abrufen lassen. Der Draft spezifiziert eine Reihe von Eigenschaften, die damit vom Client ausgelesen werden können, wie zum Beispiel „Hybrid Transport verfügbar“ oder „Discoverable Credentials möglich“. Damit würde sich der Registrierungs- und Authentifizierungsablauf an die konkreten Gegebenheiten des Systems anpassen lassen und so die Nutzererfahrung verbessern. Zum aktuellen Zeitpunkt unterstützt noch keine Plattform diese Funktion, im Chrome/Chromium-Browser ist sie aber bereits in Entwicklung [6].

Der Client wendet nun die im WebAuthn PI spezifizierte Funktion *create()* an. Sie ist eine Erweiterung des *CredentialContainer*-Objekts aus dem oben erwähnten Credential Management API. Sie kann in allen aktuellen Browsern mit *navigator.credentials.create()* aufgerufen werden. Die Funktion erhält als Argument das *PublicKeyCredentialCreationOptions*-Objekt von der Relying Party.

Die konkrete Implementierung ist plattform- und umgebungsabhängig. Im Allgemeinen fragt der Browser oder das Betriebssystem – je nachdem, ob es sich um eine Website oder eine native Anwendung handelt – den Benutzer, welchen Authentikator er verwenden möchte. Die Interaktion des Nutzers mit dem Authentikator, z. B. Fingerabdruck oder Knopf am USB-Stick, ist anschließend ebenfalls erforderlich.

Unabhängig davon, was auf Systemebene passiert, ist das Ergebnis der *create()*-Methode ein *PublicKeyCredential*-Objekt:

```
interface PublicKeyCredential : Credential {
  readonly attribute ArrayBuffer                 rawId;
  readonly attribute AuthenticatorAttestationResponse response;
  readonly attribute DOMString?                  authenticatorAttachment;
}
```

Dieses Objekt wird vom Client an den Server zurückgeschickt. Das relevante Feld ist hier die *AuthenticatorAttestationResponse*:

```
interface AuthenticatorAttestationResponse {
  readonly attribute ArrayBuffer    clientDataJSON;
  readonly attribute ArrayBuffer    attestationObject;
};
```

Bevor die Registrierung abgeschlossen werden kann, muss diese Antwort in der Relying-Party-Implementie-

Listing 2: Optionen zur Authentikatorauswahl

```
let options = {
  authenticatorSelection: AuthenticatorSelectionCriteria = {
    authenticatorAttachment: 'platform',
    residentKey: 'required',
    requireResidentKey: true,
    userVerification: 'required'
  }
};
```

Privatsphäre und Fingerprinting

Das Verheimlichen der zur Verfügung stehenden Authentikatoren geschieht natürlich nicht ohne Grund. Auch die neue Funktion wird, wenn sie den Draft verlässt, die Informationen bewusst nur andeutungsweise bereitstellen. Damit soll verhindert werden, dass Nutzer anhand ihrer Geräteeigenschaften „fingerprinted“, also über Websites hinweg nachverfolgt werden können. Es ist also wie so oft ein Spagat zwischen Nutzererfahrung und Privatsphäre.

rung validiert werden. Einerseits wird das *clientData*-Objekt überprüft, z. B. dass der Typ der Nachricht dem erwarteten *webauthn.create* entspricht, dass die Challenge dieselbe ist, die zu Beginn des Registrierungsvorgangs vom Server an den Client gesendet wurde, oder dass der Origin (der URL, von der der Client aufgerufen wurde) dem vom Server erwarteten Wert entspricht. Andererseits muss natürlich auch die Überprüfung der kryptografischen Daten im *attestationObject* stattfinden.

Dieses besteht aus zwei Teilen, dem „Attestation Statement“ und „Authenticator Data“. Das Attestation Statement stellt quasi ein Zertifikat über die Vertrauenswürdigkeit des Authentikators dar. Attestation ist ein großes Thema für sich, das in aktuellen Implementierungen oft noch außen vorgelassen wird, da es die Nutzererfahrung deutlich einschränken kann. Darauf wollen wir im nächsten Abschnitt etwas genauer eingehen. Authenticator Data enthält den eigentlichen öffentlichen Schlüssel sowie einige weitere kryptografische Daten, die vom Server validiert werden müssen.

Die meisten Bibliotheken nehmen einem die Aufgabe ab, die konkrete Validierung durchzuführen. Oft lohnt es sich aber, genauer hinzuschauen. Beispielsweise wird die Handhabung des Signature Counter in der Regel dem Benutzer der Bibliothek überlassen. Dabei handelt es sich um ein Sicherheitsfeature der WebAuthn-Spezifikation, um geklonte Authentikatoren zu erkennen.

Der Zähler ist ein Feld in der „Authentikator Data“-Datenstruktur, die Teil des *attestationObject* ist. Bei jedem Log-in-Vorgang zählt der Authentikator diesen Zähler hoch und sendet ihn an die Relying Party zurück. Das bedeutet: Um einen geklonten Authentikator zu erkennen, muss dieser Zähler serverseitig mit den Credentials gespeichert werden. Damit kann dann bei jedem Log-in-Versuch überprüft werden, ob der Zähler den erwarteten nächsten Wert hat.

Generell ist es wichtig, sich bei der Implementierung genau an die in der Spezifikation beschriebenen Schritte zu halten. Der Grundsatz „Never implement your own crypto“ sollte auch hier gelten. Aufgrund der relativen Neuheit kann es aber nicht schaden, einen genaueren Blick darauf zu werfen, was die gewählte Bibliothek eigentlich macht. Ein Fehler oder eine vergessene Validierung eines Parameters kann schnell zu einem Sicherheitsrisiko führen.

Wichtig für die Sicherheit: Attestation

Die Attestation spielt ebenfalls eine wichtige Rolle in der Sicherheitsarchitektur von FIDO2. Sie garantiert die Authentizität des Authentikators zum Zeitpunkt der Registrierung und kann dazu beitragen, Sicherheitsgarantien zu verstärken. Die Relying Party hat ein paar Möglichkeiten, die Attestation, die ein Authentikator zu leisten hat, zu beeinflussen.

In Listing 3 ist ein Ausschnitt aus dem bereits bekannten *PublicKeyCredentialCreation*-Objekt mit den Parametern zu sehen, die die Relying Party verwenden kann, um die gewünschte Art der Attestation festzulegen.

Ein Parameter ist z. B. *attestation*. Die Einstellung *direct* weist das System an, dass ein Originalnachweis der Sicherheitsmerkmale des Authentikators erforderlich ist, was eine höhere Sicherheit, aber auch mögliche Einschränkungen der Privatsphäre mit sich bringt. Hingegen bedeutet *indirect*, dass zwar ein Nachweis erforderlich ist, es aber dem Client überlassen bleibt, in welcher Form das geschieht. Dies dient vor allem dazu, den Nachweis von einer externen Trusted Party signieren zu lassen, um die Privatsphäre des Nutzers zu schützen.

Die strengste Option ist *enterprise*. Hier können eindeutige Identifikationsmerkmale enthalten sein; sie zielt vor allem auf den Einsatz in Organisationen. Am weitesten verbreitet ist nach wie vor *none* (Default-Wert), da die Einforderung einer Attestation die Nutzererfahrung negativ beeinflussen kann. Mit dem Feld *attestationFormats* kann der Server seine bevorzugten Formate angeben, in denen die Attestation dargestellt werden soll. Das ist jedoch nur ein Vorschlag; der Authentikator kann jedes beliebige Format verwenden.

Wachsam: der FIDO Metadata Service

Auf dem Markt gibt es eine Vielzahl von Authentikatoren unterschiedlicher Hersteller, die sich in ihrem Funktionsumfang deutlich unterscheiden können und für bestimmte Anwendungsfälle besser oder schlechter geeignet sind. Unternehmen oder regulierte Organisationen müssen zum Teil bestimmte Richtlinien (z. B. NIST SP800-63B für US-Bundesbehörden [7] oder BSI TR-03107 für deutsche Behörden [8]) für eine sichere Authentifizierung erfüllen, während Authentikatoren für den Konsumentenbereich sicherlich andere Sicherheitsanforderungen haben. Um sicherzustellen, dass nur vertrauenswürdige und für den Anwendungsfall geeignete Authentikatoren verwendet werden, gibt es den FIDO Metadata Service, kurz MDS [9].

MDS dient als zentrale, vertrauenswürdige Quelle für Informationen über FIDO-konforme Authentikatoren, wie z. B. deren Zertifizierungsstatus und Sicherheitseigenschaften. Konkret stellt MDS eine BLOB-Datei zur Verfügung, die unter dem URL https://mds3.fidoalliance.org heruntergeladen werden kann. Der Inhalt kann dann kopiert und z. B. unter https://jwt.io eingesehen werden.

Listing 3: PublicKeyCredentialCreationOptions

```
dictionary PublicKeyCredentialCreationOptions {
  [...]
  DOMString              attestation = "none | indirect | direct | enterprise";
  sequence<DOMString>    attestationFormats = [];
  [...]
}
```

Anhand der Informationen aus dem MDS kann die Anwendung auf bestimmte Sicherheitseigenschaften von Authentikatoren filtern, um bestimmten Richtlinien zu entsprechen. Im Konsumentenbereich gilt es aber zu bedenken, dass die Ablehnung eines Authentikators zu Verschlechterung des Nutzererlebnisses führen kann. Zielführender ist es, den Benutzer darauf hinzuweisen, dass der verwendete Authentikator nicht mehr sicher ist. Daher ist es auch wichtig, diese Datei regelmäßig zu aktualisieren, um rechtzeitig auf kompromittierte Authentikatoren reagieren zu können. Da sie sich jedoch nicht häufig ändert, empfiehlt die FIDO ein Caching der BLOB-Datei mit einem monatlichen Update [10].

Authentifizierung: Deep Dive

Beim Log-in eines registrierten Benutzers ist es seit jeher Standard, ein Formular mit der Eingabe von Benutzername und Passwort zu präsentieren. Mit Passkeys wird dies entweder auf das Feld *Benutzername* vereinfacht oder ganz weggelassen, und der Benutzer wird direkt gefragt, ob er sich mit dem für diese Website verfügbaren Passkey anmelden möchte. Voraussetzung dafür ist, dass der bei der Registrierung erzeugte Passkey vom Typ Discoverable Credentials ist.

Die Relying Party liefert ein *PublicKeyCredentialRequestOptions*-Objekt (Listing 4), wenn der Log-in-Vorgang gestartet wird. Wenn die Anwendung noch ein Feld für den Benutzernamen bereitstellt, wird dieser zu Beginn des Vorgangs an die Relying Party übergeben. Wenn diese einen Nutzer mit diesem Namen kennt, wird die ID des öffentlichen Schlüssels seines Passkeys (Credential ID) im Feld *allowCredentials* mitgesendet. Zusätzlich muss nur das Feld *challenge* mit einem zufälligen Wert gesetzt sein, die der Authentikator später mit seinem privaten Schlüssel signiert.

Die Antwort des Servers unterscheidet sich bei dem Log-in ohne Benutzerinformationen. Erhält der Server keine Benutzerinformationen, wie z. B. den Benutzernamen, so bleibt das Feld *allowCredentials* leer. Zusätzlich zur Challenge muss nun aber zwingend die ID der Relying Party im Feld *rpId* angegeben werden. In den meisten Fällen ist das einfach die Domain der Relying Party.

Die weiteren Felder sind alle optional und können analog zur Registrierung z. B. einschränken, ob und wie eine Attestation erforderlich ist.

Auf der Client-Seite wird nun *navigator.credentials.get()* mit dem *PublicKeyCredentialRequestOptions*-Objekt als Argument aufgerufen. Der CTAP2-Standard definiert die Funktion *authenticatorCredentialManagement()*, die es einem Client erlaubt, abzufragen, ob der Authentikator Anmeldedaten für eine bestimmte Relying Party besitzt. Wenn ein oder mehrere Authentikatoren mit passenden Credentials gefunden werden, sollte der Client den Benutzer fragen, welchen er verwenden möchte. Die Implementierung obliegt dem Browser- oder Plattform-Entwickler. Das bedeutet, dass die Art und Weise, wie der Benutzer gefragt wird, von Plattform zu Plattform (Chrome, Firefox, Android, iOS) unterschiedlich sein kann.

Bei der Anmeldung mit Benutzernamen ist der Ablauf ähnlich, jedoch mit einem wichtigen Unterschied: Hier sendet der Client die vom Server erhaltene Credential ID an den Authentikator. Dieser überprüft dann, ob er den zu dieser Credential ID passenden privaten Schlüssel besitzt. Handelt es sich um Non-Discoverable Log-in-Daten, wird der private Schlüssel nicht auf dem Authentikator gespeichert. Stattdessen wird er jedes Mal neu generiert, indem er aus der Credential ID und dem privaten Master Key des Authentikators abgeleitet wird. Deshalb ist in diesem Fall das Log-in mit Benutzernamen zwingend notwendig, um von der Relying Party die Credential ID des Nutzers zu erhalten.

Am Ende liefert der Authentikator in jedem Fall ein *AuthenticatorAssertionResponse*-Objekt zurück (Listing 5).

Serverseitig muss der Nutzer jetzt authentifiziert werden, indem die signierte Challenge validiert wird.

In Listing 5 ist zu sehen, dass auch ein *attestationObject* enthalten ist. Die Attestation beim Log-in ist im Standard optional und wird in einigen aktuellen Relying-Party-Implementierungen nicht direkt berücksichtigt, so auch in der Java-Bibliothek, die in der Beispielimplementierung verwendet wird. Dort ist das Feld *attestationObject* im *AuthenticatorAssertionResponse*-Objekt schlicht nicht enthalten. Sie kann dennoch in einigen Fällen sinnvoll sein, z. B. bei der Überprüfung der fortlaufenden Gültigkeit einer Attestation, wenn diese vom FIDO-Metadata-Service bereitgestellt wird.

Listing 4: PublicKeyCredentialRequestOptions

```
dictionary PublicKeyCredentialRequestOptions {
  required BufferSource                challenge;
  unsigned long                        timeout;
  USVString                            rpId;
  sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
  DOMString                            userVerification = "preferred";
  sequence<DOMString>                  hints = [];
  DOMString                            attestation = "none";
  sequence<DOMString>                  attestationFormats = [];
  AuthenticationExtensionsClientInputs extensions;
}
```

Listing 5: AuthenticatorAssertionResponse

```
interface AuthenticatorAssertionResponse : AuthenticatorResponse {
  readonly attribute ArrayBuffer      authenticatorData;
  readonly attribute ArrayBuffer      signature;
  readonly attribute ArrayBuffer?     userHandle;
  readonly attribute ArrayBuffer?     attestationObject;
};
```

Fazit

Zusammenfassend lässt sich sagen, dass bei der Implementierung und Nutzung von Passkeys viele Faktoren berücksichtigt werden müssen, von der Ablegung der Schlüssel lokal auf dem Gerät oder in der Cloud bis hin zu strengen Anforderungen an den verwendeten Authentikator: Jede Entscheidung wirkt sich auf die Sicherheit, die Benutzerfreundlichkeit und die Akzeptanz von Passkeys aus, was die Implementierung dieser vielversprechenden Technologie zu einer anspruchsvollen Aufgabe macht, bei der ständig zwischen den verschiedenen Aspekten abgewogen werden muss.

Die Unterstützung von WebAuthn in Browsern und Betriebssystemen entwickelt sich ständig weiter. Gleichzeitig wird auch die Spezifikation von der FIDO Alliance aktiv weiterentwickelt. Eine effektive Nutzung von FIDO2 erfordert daher die Berücksichtigung der aktuellen Unterstützungsniveaus und möglicherweise die Implementierung von Fallback-Methoden für Systeme, die den Standard noch nicht unterstützen.

Unser Fazit: Richtig umgesetzt bieten Passkeys für den Endnutzer hinsichtlich Benutzererfahrung und Sicherheit einen deutlichen Mehrwert, der Passwörter ablösen kann. Mittlerweile bieten fast alle großen Technologieplattformen Passkeys als Alternative zum klassischen Passwort an. Im Rahmen dieses Artikels haben wir auch eine Demoanwendung entwickelt, in der man sich durch den Registrierungs- und Log-in-Prozess klicken kann. In den jeweiligen Teilschritten der Flows kann man auch sehen, welche Daten hin- und hergeschickt werden. Das Repository ist auf GitHub [11] eingecheckt. Dort findet ihr auch eine Anleitung, wie ihr den Code lokal starten könnt.

Christoph Eifert ist seit fast vier Jahren als agiler Softwareentwickler bei der andrena objects ag tätig. Als neugieriger Fullstack-Entwickler möchte er immer die ganze Software im Blick haben. Er begeistert sich für die Theorie neuer Netzwerkprotokolle und Hardware-Level-Schwachstellen ebenso wie für Secure-Coding-Praktiken und Vulnerability Testing.

Jens Le arbeitet seit drei Jahren als agiler Softwareentwickler bei der andrena objects ag. Neben der Entwicklung von Anwendungen mit Java/JakartaEE und Typescript konnte er bereits Erfahrungen beim Thema Authentifikation in unterschiedlichen Projekten sammeln.

Links & Literatur

[1] https://www.w3.org/TR/webauthn-2/

[2] "https://fidoalliance.org/specs/fido-v2.2-rd-20230321/fido-client-to-authenticator-protocol-v2.2-rd-20230321.html

[3] https://w3c.github.io/webappsec-credential-management

[4] https://html.spec.whatwg.org/multipage/system-state.html

[5] https://w3c.github.io/webauthn/

[6] https://cr-status.appspot.com/feature/5128205875544064

[7] https://pages.nist.gov/800-63-3/sp800-63b.html

[8] https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/TR-nach-Thema-sortiert/tr03107/TR-03107_node.html

[9] https://fidoalliance.org/specs/mds/fido-metadata-statement-v3.0-ps-20210518.html

[10] https://fidoalliance.org/metadata/

[11] https://github.com/andrena/java-magazin-passkeys