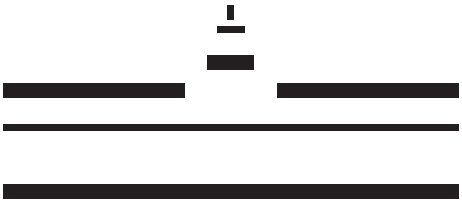


Westfälische
Wilhelms-Universität
Münster

secureFIT

–

Mehrfaktor-Authentifizierung und Autorisierung

Bachelorarbeit
zur Erlangung des akademischen Grades
Bachelor of Science

Vorgelegt von:
Tim Düsterhus

Thema gestellt von:
Dr. Dietmar Lammers

Arbeit betreut durch:
Dr. Dietmar Lammers

Sassenberg, 12. Dezember 2016

Abstract

Diese Arbeit stellt ein Konzept für eine sichere Zugriffsverwaltung in einem zentral verwalteten Mehrbenutzersystem mit unterschiedlichen Kompetenzen einzelner Nutzer vor. Dazu werden zwei standardisierte Verfahren zur Mehrfaktor-Authentifizierung im Detail untersucht und vorgestellt, wie dadurch der Zugriff auf Konten einzelner Nutzer besonders geschützt werden kann. Weiterhin wird untersucht, wie nach erfolgter Authentifizierung des Nutzers sichergestellt werden kann, dass jeweils nur berechtigte Nutzer die einzelnen Funktionen des Systems nutzen können.

Viele Systeme besitzen bereits eine bestehende Einfaktor-Authentifizierung und sind nicht von Grund auf mit dem Konzept einer Zweifaktor-Authentifizierung entwickelt worden. Daher zeigt diese Arbeit anhand einer beispielhaften Implementation eines der vorgestellten Verfahren, wie man ein derartiges Zweifaktor-System nachträglich in seine bestehende Authentifizierung integrieren kann, ohne dabei große Änderungen an den bereits erprobten Komponenten vornehmen zu müssen.

Inhaltsverzeichnis

1. Einleitung

Kennwörter und Pins sind aus dem Leben des 21. Jahrhunderts nicht mehr wegzudenken. Bevor das Internet massentauglich wurde, besaß der Durchschnittsbürger seine EC-Karte, SIM-Karte und möglicherweise einige wenige Kennwörter. Heutzutage benötigt man für jeden Online-Shop, jedes soziale Netzwerk, für seine E-Mail-Adressen und für die zentrale Nutzerkennung an der Universität ein gesondertes Benutzerkonto. Und für jedes dieser Konten sollte man ein getrenntes Kennwort wählen. Die Realität sieht aber anders aus: Viele Menschen besitzen nur ein einziges Kennwort, das sie für jedes ihrer Benutzerkonten verwenden. Sobald ein Angreifer an ebenjenes Kennwort gelangt ist, stehen ihm alle Türen zum digitalen Leben des Betroffenen offen.

Ein Angreifer kann über eine Vielzahl von Möglichkeiten an Kennwörter gelangen. Eine simple Möglichkeit ist es einfach zu fragen: Viele Menschen verraten ihr Passwort für eine einfache Gegenleistung [Uni16]. Eine andere Möglichkeit sind Sicherheitslücken in der Software von Online-Diensten. So listet die Internetseite ‚`';--have i been pwned?`‘ beispielsweise rund 1,8 Milliarden Nutzerkonten auf 152 unterschiedlichen Portalen, deren private Daten ungewollt an die Öffentlichkeit gelangt sind [HIBP]. Beide dieser Methoden stützen sich darauf, dass der Betroffene leichtfertig mit seinen Kennwörtern umgegangen ist, aber auch technisch versierte Menschen können beispielsweise durch Schadsoftware von einem Kennwort-Diebstahl betroffen sein.

Es ist also leicht zu sehen, dass nur Kennwörter nicht ausreichen, um wichtige Benutzerkonten zu schützen. Immer mehr Dienste sind sich dieser Tatsache bewusst und bieten daher einen Zweifaktor-Login an. Wenn ein Nutzer sein Konto mit dieser Funktion speziell schützt, dann ist das Kennwort allein nicht mehr ausreichend, um in das Konto und somit an die persönlichen Daten des Nutzers zu gelangen. Stattdessen muss beispielsweise ein über SMS empfangener Code oder ein Einmalkennwort von einer zuvor erhaltenen Liste eingegeben werden. Bekannt ist dieses Verfahren vom Online-Banking: Neben der PIN ist

für jede Transaktion zusätzlich eine neue Transaktionsnummer (TAN) erforderlich.

Die sichere Authentifizierung des Benutzers ist jedoch nicht ausreichend, um sicherzustellen, dass der Nutzer lediglich auf Informationen und Funktionen Zugriff erhält, die für ihn bestimmt sind. Es muss beispielsweise sichergestellt sein, dass ein Benutzer nicht in der Lage ist Informationen anderer Nutzer zu verändern oder einzusehen, ohne dafür speziell von einem Administrator berechtigt worden zu sein. Im Gegensatz zu fehlerhafter Authentifizierung sind Probleme in der Autorisierung in der Regel schwieriger zu erkennen: Die erfolgreiche Authentifizierung mit falschem Kennwort wird schnell auffallen, die versehentliche Veröffentlichung von sensiblen Informationen könnte hingegen an vielen Stellen passieren. Der Dienst GitHub versendete beispielsweise Benachrichtigungs-E-Mails mit einem privatem Link zum Abbestellen dieser Benachrichtigungen. Beim Antworten auf diese E-Mails war es möglich, dass dieser Link in einem Zitat innerhalb der Antwort enthalten war und dadurch veröffentlicht wurde. Die Problematik wurde so korrigiert, dass der Link in Zukunft nur noch dem zugeordneten Nutzer zugänglich war [Git16].

Zur Lösung dieses Problems soll in dieser Arbeit ein Konzept zur sicheren Integration eines Zweifaktor-Logins in ein bestehendes Authentifizierungssystem vorgestellt und zusätzlich auf technischer und administrativer Basis diskutiert werden, wie die sicher authentifizierten Nutzer für die unterschiedlichen Funktionen der Anwendung autorisiert werden können, sodass die Wahrscheinlichkeit für eine Fehlkonfiguration minimiert wird.

Dazu werden in Kapitel 2 Universal Second Factor (U2F) und Time-based One-time Password Algorithm (TOTP) als Repräsentanten von Smartcard- beziehungsweise Einmalkennwort-basierten Verfahren im Detail vorgestellt und anschließend in Kapitel 3 untersucht, wie das U2F-Verfahren in die bestehende Anwendung integriert werden kann. Nachdem die sichere Authentifizierung gewährleistet wurde, behandelt Kapitel 4 die sichere Autorisierung und stellt unterschiedliche Autorisierungsstrategien mit ihren jeweiligen Vor- und Nachteilen vor. Kapitel 5 schließt mit einem übergreifenden Fazit und gibt zugleich Anstöße für weitere sicherheitsrelevante Überlegungen, die als Ausgangspunkte für nachfolgende Untersuchungen herangezogen werden können.

2. Zweifaktor-Authentifizierung

2.1. Einführung

Es gibt unterschiedliche Möglichkeiten einen Nutzer eindeutig zu identifizieren, allen Verfahren ist jedoch gemein, dass sie sich in eine von drei Kategorien einordnen lassen [Fed, Seite 3]:

Wissen Kennwörter, Private Informationen

Besitz TAN-Liste, Kryptografische Schlüssel, Smartcards

Biometrie Fingerabdruck, Iris-Scan

Die Verfahren einer Kategorie teilen sich dabei aufgrund ihrer inhärenten Eigenschaften die grundsätzlichen Vor- und Nachteile. So ist es beispielsweise trivial möglich, *Wissen* weiterzugeben und damit das Identifizierungsmerkmal zu duplizieren. Es ist jedoch nahezu unmöglich, die Identifizierung durch ein *Biometrisches Merkmal* an eine andere Person weiterzugeben. Dadurch ist die Authentifizierung durch ein Biometrisches Merkmal, unter der Annahme eines perfekt arbeitenden Systems - also eines Systems, das keine Fehler bei der Erkennung des Merkmals macht -, sicherer als die Authentifizierung über ein Kennwort.

Im Gegenzug ist es nicht möglich, ein *Biometrisches Merkmal* im Falle einer Kompromittierung zu verändern, während dies beispielsweise bei einem Kennwort sehr einfach möglich ist.

Verfahren, die auf dem *Besitz* von etwas basieren, gehen in vielen Eigenschaften einen Mittelweg zwischen Wissen und Biometrie. Eine Smartcard ist beispielsweise schwierig zu duplizieren, kann aber weiterhin an andere Personen weitergegeben werden. Der ursprüngliche Besitzer verliert dabei jedoch die Möglichkeit sich zu authentifizieren. Auch ist die Integration eines derartigen Systems mit höheren Kosten verbunden als die einfache Abfrage von Wissen,

beispielsweise muss für jeden Mitarbeiter eine Smartcard beschafft werden, die Kosten sind jedoch im Allgemeinen geringer als die für ein sicheres Biometrisches System.

Eine sichere Authentifizierung kombiniert also Verfahren aus unterschiedlichen Kategorien, damit die jeweiligen Nachteile eines Verfahrens durch ein anderes Verfahren ausgeglichen werden können. Die Authentifizierung auf Basis von *Wissen* in Form von Kennwörtern ist die am weitesten verbreitete, da diese am einfachsten implementiert werden kann. Daher werden nachfolgend zwei auf *Besitz* basierende Verfahren vorgestellt. In Kombination mit Kennwörtern kommen dann Verfahren aus zwei Kategorien zum Einsatz. Das U2F-Verfahren (Kapitel 2.2) ist ein Smartcard-basiertes Verfahren, der Benutzer authentifiziert sich über den Besitz eines USB-Sticks. Das TOTP-Verfahren (Kapitel 2.3) basiert auf kryptografischen Schlüsseln; der Benutzer authentifiziert sich über einen auf seinem Smartphone gespeicherten zufällig generierten Schlüssel, mithilfe dessen Einmalkennwörter generiert werden.

2.2. Universal Second Factor

Universal Second Factor (U2F) ist ein von der FIDO Alliance Ende 2014 veröffentlichtes offenes Verfahren zur Authentifizierung mittels einer Smartcard. Ziel war es, ein einheitliches und sicheres Verfahren zur Zweifaktor-Authentifizierung zu schaffen [FID14], um die Akzeptanz für Zweifaktor-Authentifizierung zu erhöhen. Mit dem Webbrowser Google Chrome ist das Verfahren bereits nativ in einem der führenden Webbrowser integriert, für Mozilla Firefox gibt es ein Add-on [Chm]. Weiterhin gibt es bereits eine Vielzahl kompatibler Smartcards unterschiedlicher Hersteller, unter anderem stand bereits einen Monat nach Veröffentlichung der Spezifikation ein Iris-Scanner zur Verfügung [Eye15].

2.2.1. Voraussetzungen

Auf Seiten des Anbieters ist es notwendig, dass eine Bibliothek mit Unterstützung für Elliptic Curve Digital Signature Algorithm (ECDSA)-Signaturen auf der durch das National Institute of Standards and Technology (NIST) standardisierten P-256-Kurve [Nat13, Seite 100] zur Validierung der durch die Smartcard übermittelten Daten zur Verfügung steht (eine verbreitete Bibliothek ist

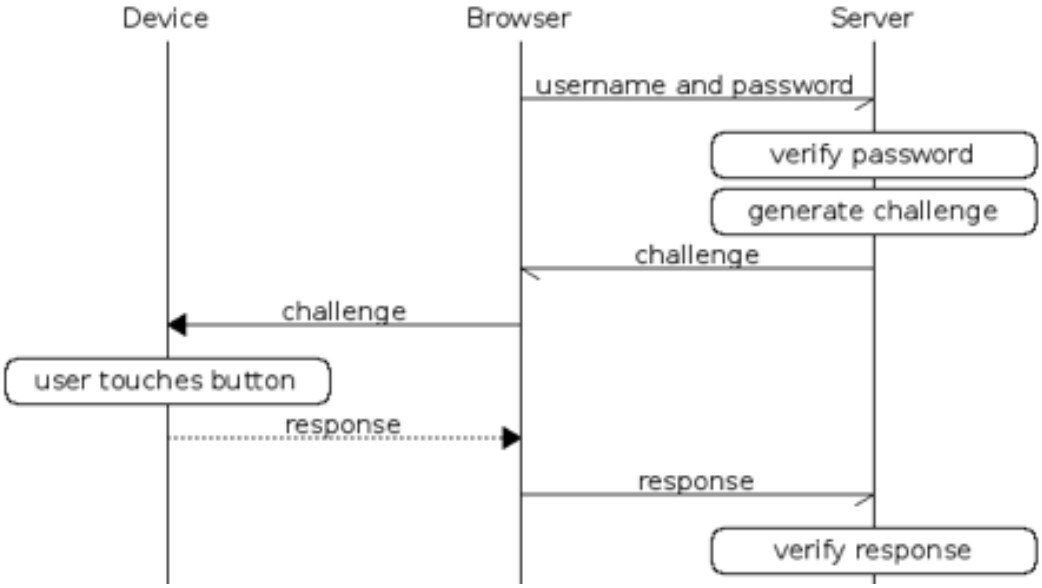


Abbildung 2.1.: Ablauf der Authentifizierung mit U2F [Yub].

OpenSSL)[1]. Der Benutzer benötigt eine Smartcard, die das U2F-Verfahren unterstützt, sowie einen kompatiblen Webbrowser.

2.2.2. Funktionsweise

Die Internetseite des Anbieters kommuniziert über ein vom Webbrowser zur Verfügung gestelltes JavaScript-API mit der Smartcard. Dieses API stellt zwei Funktionen zur Verfügung [BBL15, Abschnitt 3.1.1]:

u2f_register_request zum Registrieren der Smartcard im Konto eines Nutzers.

u2f_sign_request zum Authentifizieren eines Nutzers.

Beide Funktionen arbeiten nach dem Challenge-Response-Verfahren (Abbildung 2.1). Der Anbieter generiert eine Challenge, welche einen zufälligen Wert (Nonce) sowie weitere Parameter enthält, und sendet diese über das API an die Smartcard. Nachdem der Benutzer die Anfrage bestätigt hat (beispielsweise durch Drücken einer physischen Taste an der Smartcard), sendet die Smartcard eine signierte Antwort zurück an den Webbrowser, die von diesem über das API der Anwendung zur Verfügung gestellt wird.

[1] Theoretisch ist es möglich diese Validierung selbst zu implementieren. Die Gefahr für Fehler und dadurch induzierte Sicherheitslücken wäre jedoch immens.

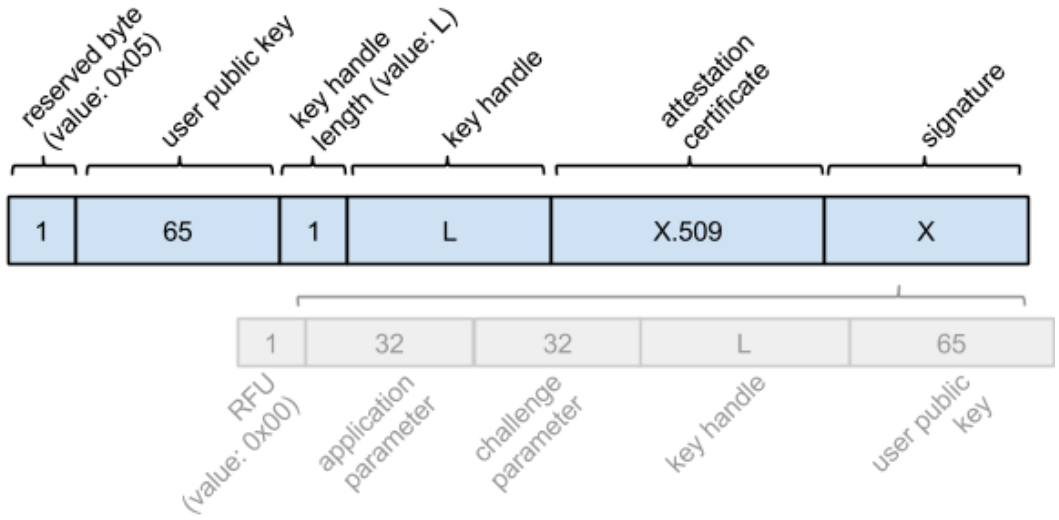


Abbildung 2.2.: Aufbau der Antwort auf einen `u2f_register_request` [BE15, Abschnitt 4.3].

Bei jeder dieser Anfragen erwartet die Smartcard eine `appId`. Diese trennt den Speicher der Smartcard in unterschiedliche Namensräume: Die zuvor registrierten Daten sind beim Authentifizieren nur dann sicht- und benutzbar, wenn die `appId` beider Anfragen übereinstimmt. Der zulässige Inhalt der `appId` unterliegt einem komplexen Regelwerk [BH15, Abschnitt 3.1.2], welches sicherstellen soll, dass ein bösartiger Dienst nicht in der Lage ist die `appId` (und somit die Schlüssel) eines anderen Dienstes zu nutzen. Vereinfacht lässt sich sagen, dass die `appId` von der in der Adressleiste des Webbrowsers sichtbaren Domain abgeleitet sein muss.

Einrichtung

Um eine neue Smartcard mit dem Konto eines Nutzers zu verknüpfen, sendet der Dienst eine Anfrage des Typs `u2f_register_request` an die Smartcard. Neben der `appId` und der Nonce besteht die Möglichkeit etwaige bereits bekannte Schlüssel in der Anfrage zu vermerken. Dadurch ist es der Smartcard möglich zu erkennen, dass sie bereits mit dem Konto des Nutzers verknüpft ist, um dadurch zu vermeiden, dass eine Smartcard mehrfach mit dem identischen Konto verknüpft wird [BBL15, Abschnitt 5.1.3].

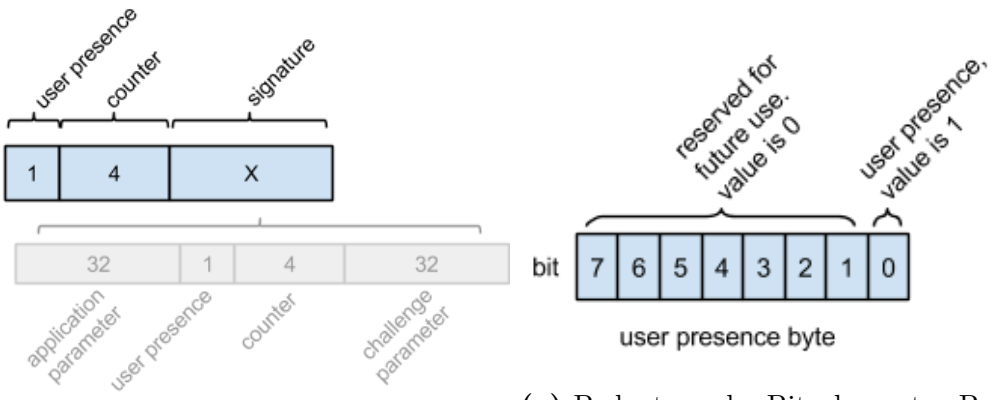
Nachdem der Nutzer die Anfrage durch Drücken der Taste an der Smartcard bestätigt hat, generiert die Smartcard ein neues ECDSA-Schlüsselpaar. Der öffentliche Schlüssel des Schlüsselpaares wird zusammen mit einem Bezeichner für dieses Schlüsselpaar, einem Attestierungszertifikat und einer ECDSA-Signatur eines SHA-256-Hashs zurück an den Webbrowser gesendet (Abbildung 2.2).

Diese Antwort ist vom Dienst in ihre Bestandteile zu zerlegen und anschließend zu überprüfen: Zum einen müssen alle Felder der Antwort das spezifizierte Format aufweisen. So ist unter anderem zu prüfen, dass das reservierte Byte am Anfang der Antwort den Wert `0x05` enthält. Zum anderen ist die angehängte Signatur zu prüfen. Diese muss vom mitgesendeten Attestierungszertifikat stammen und sichert neben den einzelnen Feldern der Antwort auch die ursprüngliche Anfrage an die Smartcard. Dadurch ist sichergestellt, dass die Antwort „frisch“ auf Basis der Anfrage erzeugt und nicht im Rahmen eines Replay-Angriffs untergeschoben wurde. Das Attestierungszertifikat soll hingegen identisch auf einer Vielzahl an Smartcards eines Herstellers hinterlegt sein und es dem Dienst somit ermöglichen, nur speziell autorisierte Smartcards zu erlauben, um dadurch ein gewisses Sicherheitsniveau zu garantieren[2] [Sri+15, Abschnitt 8]. Beim Parsen der Antwort ergibt sich die Schwierigkeit, dass die Länge des Attestierungszertifikats nicht explizit angegeben ist. Stattdessen muss die Struktur des X.509-Zertifikats [RFC5280, Abschnitt 4.1] nach den in X.690[3] spezifierten Distinguished Encoding Rules (DER) für die Abstract Syntax Notation One (ASN.1) analysiert und dadurch die Länge des Zertifikats ermittelt werden [BE15, Abschnitt 4.3].

Nachdem der Anbieter die Antwort der Smartcard verifiziert hat, ist es noch erforderlich den erhaltenen öffentlichen Schlüssel und Bezeichner zu hinterlegen, damit er für die Authentifizierung des Nutzers genutzt werden kann.

[2] Keine der Smartcards, die dem Autor dieser Arbeit zum Testen zur Verfügung standen, implementierte das Attestierungszertifikat konform zur Spezifikation. Eine Smartcard erzeugte für jeden Schlüssel ein neues Attestierungszertifikat, die andere verwendete ein Attestierungszertifikat, welches die Seriennummer der Smartcard enthielt und es somit erlaubt, die Smartcard über Dienste hinweg eindeutig zu identifizieren.

[3] International Telecommunication Union. *Information Technology — ASN.1 Encoding Rules — Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER).* ITU-T Recommendation X.690. Juli 2002. URL: `http://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf`.



(a) Bedeutung der Bits des ersten Bytes.

Abbildung 2.3.: Aufbau der Antwort auf einen `u2f_sign_request` [BE15, Abschnitt 5.4].

Authentifizierung

Um einen Nutzer nach erfolgter Einrichtung zu authentifizieren, wird eine Anfrage des Typs `u2f_sign_request` an die Smartcard übermittelt. Diese enthält die `appId`, die Nonce und die bei der Einrichtung gespeicherten Bezeichner der zur Authentifizierung zulässigen Schlüsselpaare.

Nachdem der Nutzer die Anfrage durch Drücken der Taste an der Smartcard bestätigt hat, sendet die Smartcard analog zur Registrierung signierte Daten (Abbildung 2.3) zurück an den Webbrowser. Anders als bei der Registrierung, sind die Daten nicht mit dem Attestierungszertifikat, sondern mit dem privaten Schlüssel eines der bei der Anfrage genannten Schlüsselpaare signiert. Der Bezeichner des verwendeten Schlüssels wird getrennt von den signierten Daten an den Webbrowser übermittelt.

Um die Signatur zu prüfen, ist es erforderlich den bei der Registrierung erhaltenen öffentlichen Schlüssel zu vervollständigen, damit der Schlüssel durch übliche Kryptografie-Bibliotheken verarbeitet werden kann. Der Grund dafür ist, dass die bei der Einrichtung übermittelten Daten lediglich den Punkt auf der NIST P-256-Kurve, ohne weitere Informationen, repräsentieren. Im Allgemeinen ist es dazu erforderlich den Kurvenpunkt in die `SubjectPublicKeyInfo`-Struktur eines X.509-Zertifikats einzubetten, damit die verwendete Bibliothek erkennt, dass es sich um einen Punkt auf der P-256-Kurve und nicht etwa um einen RSA-Modulus und -Exponenten handelt. Der ASN.1-Aufbau

der `SubjectPublicKeyInfo`-Struktur für ECDSA-Schlüssel ist in RFC 5480[4] beschrieben.

Neben der Validierung der Signatur hat der Anbieter zu prüfen, dass der in der Antwort übermittelte `counter`-Wert streng monoton ansteigt. Sollte der übermittelte Wert geringer als der zuletzt bekannte sein, so wurde die Smartcard möglicherweise kopiert oder modifiziert und somit kompromittiert. Die Authentifizierung des Nutzers kann in diesem Fall natürlich nicht erfolgen.

Nachdem der Benutzer erfolgreich authentifiziert, also die Signatur validiert und der `counter`-Wert überprüft wurde, ist noch der übermittelte `counter`-Wert zu hinterlegen, um das Verfahren abzuschließen.

2.2.3. Analyse des Verfahrens

Universal Second Factor setzt bei der Kommunikation zwischen Smartcard und Dienst konsequent auf signierte Nachrichten, wobei das Verfahren auf die durch NIST zertifizierte P-256-Kurve standardisiert wurde. Dieses Verfahren ist unter Kryptografen umstritten. Einerseits, da einige Designentscheidungen der Kurve unklar sind, andererseits, da es unverhältnismäßig schwierig ist dieses Verfahren vollständig korrekt zu implementieren [BL14]. Dennoch gilt das Verfahren grundsätzlich als sicher, daher soll der Fokus dieser Analyse auf der praktischen Umsetzung des Verfahrens und anderen Designentscheidungen liegen.

Die Sicherheit von U2F basiert auf der Sicherheit der ECDSA-Signaturen. Das primäre Standbein ist also die Geheimhaltung der privaten Schlüssel. Dies beginnt bereits bei der Erzeugung des Schlüsselpaares. Es ist essentiell, dass die Smartcard einen sicheren, kryptografischen Zufallsgenerator enthält, da die privaten Schlüssel andernfalls unter Umständen berechnet oder erraten werden könnten. Nachdem die Schlüssel erzeugt wurden, dürfen diese die Smartcard nicht verlassen. Die einzig zulässige Operation ist es, die Smartcard darum zu bitten eine Authentifizierungsanfrage zu signieren. Neben der Geheimhaltung der Schlüssel stellt die Spezifikation weitere Anforderungen an den Hersteller der Smartcard, denn so muss zum Beispiel die Trennung der Namensräume auf Basis der `appId` korrekt umgesetzt sein. Auch muss sichergestellt werden, dass

[4] S. Turner u. a. *Elliptic Curve Cryptography Subject Public Key Information.* RFC 5480. RFC Editor, März 2009. URL: `http://www.rfc-editor.org/rfc/rfc5480.txt`.

eine Nutzung der Smartcard durch den Webbrowser nur möglich ist, wenn der Taster an der Smartcard betätigt wurde.

Auf Seiten der Webbrowser-Hersteller muss sichergestellt sein, dass ein Anbieter mit bösartigen Absichten nicht in der Lage ist die `appId` eines anderen Anbieters zu nutzen. Dazu muss das in Abschnitt 2.2.2 bereits genannte Regelwerk vollständig korrekt implementiert werden [BH15, Abschnitt 3.1.2].

Schlussendlich muss auch der Anbieter das Verfahren korrekt implementieren: Er muss die ECDSA-Signaturen korrekt prüfen und sicherstellen, dass der `counter`-Wert streng monoton ansteigt.

Auch wenn das U2F-Verfahren in der Theorie sicher sein mag, ist es ein sehr komplexes Verfahren, denn es müssen drei unterschiedliche Parteien jeweils alle für sie relevanten Details der Spezifikation vollständig korrekt implementieren. Wenn nur eine der Parteien einen Fehler in der Umsetzung macht, ist die Sicherheit gefährdet. Für den Benutzer ist es schwierig zu prüfen, ob alle Teile korrekt funktionieren. So ist es nahezu unmöglich zu prüfen, ob der Zufallsgenerator innerhalb der Smartcard zuverlässig funktioniert. Ebenso ist es ohne Programmiererfahrung nicht möglich zu prüfen, ob der Webbrowser die Validierung der `appId` korrekt umsetzt.

Diese Komplexität schützt aber gegen Probleme, gegen die beispielsweise das TOTP-Verfahren nicht schützen kann (Abschnitt 2.3.3): Bei korrekter Umsetzung des U2F-Verfahrens ist der Benutzer gegen Phishing-Angriffe geschützt. Aufgrund der `appId` ist es für eine Phishing-Seite nicht möglich eine gültige Signatur der Challenge des Anbieters zu erwirken. Selbst wenn der Phisher in der Lage ist durch einen Man in the Middle-Angriff die Domain des Anbieters zu kapern, kann der Phishing-Angriff abgewehrt werden. Es ist nämlich möglich, gewisse Parameter der Transport Layer Security (TLS)-Verbindung mit in die Challenge einzubetten [Sri+15, Seite 8]. Diese Parameter können durch den Phisher nicht kontrolliert werden und unterscheiden sich daher bei der Verbindung zwischen Nutzer und Phisher sowie Phisher und Anbieter, wodurch der Man in the Middle-Angriff auffliegt. Auch ist es schwierig das Authentifizierungsmerkmal zu duplizieren. Selbst wenn ein Angreifer an das Schlüsselpaar gelangen würde, müsste er einen plausiblen Wert für den streng monotonen `counter` erraten. Wählt er zu niedrig, dann fällt der Angriff sofort auf. Wählt er deutlich zu hoch, dann ist es ebenfalls unwahrscheinlich, dass die ursprüngliche Smartcard die Daten signiert hat.

Fazit

U2F ist ein sicheres Verfahren, bei dem im Entwurf eine Vielzahl von Angriffsmöglichkeiten bedacht und abgewehrt wurden. Diese Sicherheit stützt sich jedoch auf die korrekte Implementation der umfangreichen Spezifikation. Wie in Fußnote 2 angemerkt, implementiert keine der Smartcards des Autors das Attestierungszertifikat korrekt. Dieser Fehler in der Umsetzung ist zwar nicht sicherheitsrelevant, jedoch im Falle der einen Smartcard aus Gründen der Privatsphäre fragwürdig. Dies macht deutlich, dass Fehler in der Umsetzung definitiv vorkommen und dadurch die Sicherheit von Universal Second Factor gefährden.

2.2.4. Vergleich mit anderen Smartcard-Verfahren

Es gibt eine Vielzahl von Unternehmen, die Smartcards für die Nutzung am Rechner anbieten. Die meisten dieser Systeme sind jedoch nicht unmittelbar zur Nutzung im Bereich der Mehrfaktorauthentifizierung geeignet. Stattdessen sind diese dazu gedacht, die Verwendung von Software im Bereich der asymmetrischen Verschlüsselung durch eine Hardware-basierte Lösung zu ersetzen. Beispielsweise lassen sich ein Großteil der angebotenen Smartcards mithilfe von Pretty Good Privacy (PGP)-Software ansteuern. Die Smartcards, welche für die Verwendung als Authentifizierungsmerkmal konzipiert wurden, sind jedoch oftmals proprietär. So wird beispielsweise ein USB-Stick zum sicheren Versenden seiner Steuererklärung mittels ELSTER angeboten [sec]. Dieser USB-Stick ist allerdings nicht für andere Zwecke verwendbar. Breitere Unterstützung haben Systeme von Yubico und Nitrokey, allerdings ist man auch hier bei den älteren Modellen an den Hersteller gebunden und von diesem abhängig. Beide Unternehmen bieten jedoch mittlerweile auch Smartcards an, die neben dem proprietären Verfahren ebenso U2F unterstützen.

Fazit

Universal Second Factor bietet den großen Vorteil, dass es ein offener Standard ist. Dadurch besteht weder für den Anbieter, noch für den Nutzer die Gefahr, dass das System eingestellt wird und ein neues Verfahren implementiert beziehungsweise eine neue Smartcard erworben werden muss. Durch die standardmäßige Integration in Webbrowsern und Hersteller, die einfache U2F-

kompatible Smartcards bereits für unter 10 € anbieten, ist die Einstiegshürde sehr gering. So müssen keine Geräte-Treiber installiert werden und die Investition lohnt sich auch dann, wenn noch nicht viele der genutzten Dienste das Verfahren implementieren.

2.3. Time-based One-time Password Algorithm

Der Time-based One-time Password Algorithm (TOTP), auch bekannt als RFC 6238[5], ist eine Erweiterung des in RFC 4226[6] spezifizierten HMAC-Based One-Time Password Algorithm (HOTP). Durch die Standardisierung als RFC und einfacher Implementierung auf Seiten des Dienstes, ist es das wohl beliebteste Verfahren für Zweifaktor-Authentifizierung im Web. Die Google Authenticator-Client-Implementierung (Abbildung 2.4) für Android verzeichnet beispielsweise zwischen 10 und 50-Millionen Installationen und rund 140-Tausend Bewertungen [Goo].

2.3.1. Voraussetzungen

Die Anforderungen auf Seiten des Anbieters beschränken sich auf eine korrekte Systemzeit. Auf Seiten des Nutzers ist ein TOTP-Generator erforderlich. Im Regelfall ist dies eine App für das eigene Smartphone. Der Generator könnte aber auch in den eigenen Passwort-Safe integriert sein [Mar].

2.3.2. Funktionsweise

Einrichtung

Bei der Einrichtung der Zweifaktor-Authentifizierung mittels TOTP generiert der Dienst mithilfe eines kryptografisch sicheren Zufallszahlengenerators ein Shared Secret, welches der Benutzer in seine TOTP-Anwendung überträgt. Um die Übertragung zu erleichtern, bieten übliche Client-Anwendungen die Möglichkeit, das Shared Secret zusammen mit dem Namen des Dienstes und den Parametern des Algorithmus in Form eines QR-Code an eine Smartphone-Anwendung zu übertragen (Abbildung 2.5) [Goo15].

[5] D. M'Raihi u. a. *TOTP: Time-Based One-Time Password Algorithm*. RFC 6238. RFC Editor, Mai 2011. URL: http://www.rfc-editor.org/rfc/rfc6238.txt.

[6] D. M'Raihi u. a. *HOTP: An HMAC-Based One-Time Password Algorithm*. RFC 4226. RFC Editor, Dez. 2005. URL: http://www.rfc-editor.org/rfc/rfc4226.txt.

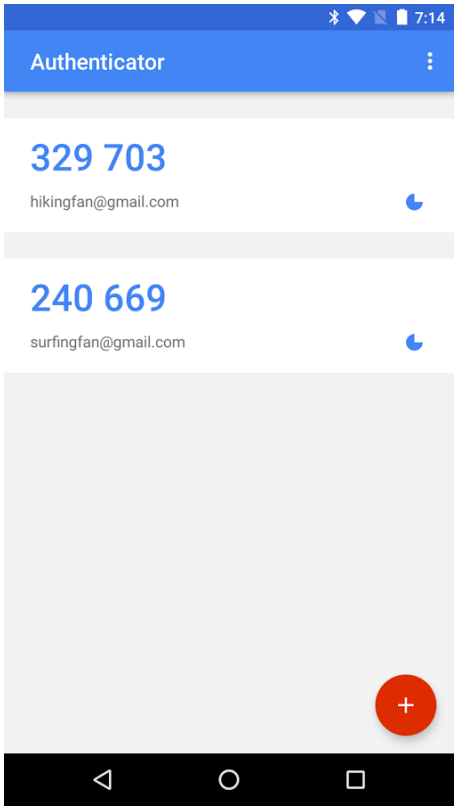


Abbildung 2.4.: Ansicht der Einmalkennwörter in Google Authenticator für Android [Goo].

Abbildung 2.5.: QR-Code, welcher das TOTP-Shared Secret enthält.

Authentifizierung

Sowohl dem Anbieter als auch der Client-Anwendung sind die Parameter des Algorithmus sowie das vereinbarte Shared Secret bekannt. Bei der Authentifizierung bittet der Benutzer seine Client-Anwendung das aktuelle Einmalkennwort zu erzeugen und sendet es an den Anbieter. Dieser generiert nun ebenfalls das aktuelle Einmalkennwort und vergleicht es mit dem übermittelten Einmalkennwort des Nutzers. Wenn beide Kennwörter übereinstimmen, dann ist der Nutzer erfolgreich authentifiziert worden.

Der Algorithmus zur Generierung der Einmalkennwörter mittels HOTP (der Basis für TOTP) lautet:

Algorithmus 2.1 HMAC-based One-time Password Algorithm

$H \leftarrow$ HMAC-SHA1(K, C) {$H[0..159]$ ist nun ein 160-Bit String}
$O \leftarrow H[156..159]$ {O ist die Zahl, die durch die letzten 4 Bit von H repräsentiert wird}
$S \leftarrow H[O \times 8 + 1..O \times 8 + 31]$ {S ist die Zahl, die, ohne Berücksichtigung des Most Significant Bit, durch die Bytes O bis $O + 3$ von H repräsentiert wird}
return $S \mod 10^D$

Hierbei bezeichnet HMAC-SHA1 den in RFC 2104[7] beschriebenen Algorithmus unter Verwendung des in RFC 3174[8] definierten Secure Hash Algorithm 1, K das Shared Secret, C einen streng monoton steigenden Zähler und D die Länge des gewünschten Einmalkennwortes.

TOTP definiert den Wert von C wie folgt auf Basis der aktuellen Uhrzeit:

Algorithmus 2.2 Time-based One-time Password Algorithm

return $\left\lfloor \frac{T}{X} \right\rfloor$

Hierbei bezeichnet T die Anzahl der Sekunden, die seit dem 1. Januar 1970 00:00:00 UTC vergangen sind (UNIX-Timestamp) und X die mit der Client-Anwendung vereinbarte Schrittgröße.

Unter Verwendung der Standardparameter ($X = 30$, $D = 6$) wird also alle 30 Sekunden ein neues Einmalkennwort, bestehend aus 6 Ziffern, erzeugt.

[7] Hugo Krawczyk, Mihir Bellare und Ran Canetti. *HMAC: Keyed-Hashing for Message Authentication*. RFC 2104. RFC Editor, Feb. 1997. URL: `http://www.rfc-editor.org/rfc/rfc2104.txt`.

[8] D. Eastlake und P. Jones. *US Secure Hash Algorithm 1 (SHA1)*. RFC 3174. RFC Editor, Sep. 2001. URL: `http://www.rfc-editor.org/rfc/rfc3174.txt`.

Um die Benutzerfreundlichkeit zu erhöhen, ist es möglich, dass der Anbieter zusätzlich das jeweils vorherige und nächste Einmalkennwort akzeptiert. Dazu inkrementiert beziehungsweise dekrementiert der Anbieter den mittels Algorithmus 2.2 ermittelten Wert von C um 1. Dies erlaubt es dem Benutzer sich zu authentifizieren, auch wenn die Uhrzeit seiner Client-Anwendung leicht abweicht oder er den Code erst am Ende des 30-sekündigen Zeitfensters abliest.

2.3.3. Analyse des Verfahrens

Primärer Baustein des TOTP-Verfahrens ist der Keyed-Hash Message Authentication Code (HMAC), welcher gleich zu Beginn berechnet wird (Algorithmus 2.1). Die Sicherheit des Verfahrens steht und fällt mit der Sicherheit der HMAC. Konkret müssen zwei Eigenschaften von der HMAC gefordert werden:

1. Die Ausgabe darf ohne Kenntnis des Shared Secrets nicht vorhersehbar sein.

2. Die Ausgabe muss zufällig verteilt sein.

Wird die erste Eigenschaft verletzt, so ist das Verfahren offensichtlich unsicher. Wird die zweite Eigenschaft verletzt, so wäre es möglich taktisch an einen Brute-Force-Angriff heranzugehen und somit die Chance zu erhöhen ein richtiges Einmalkennwort zu erraten.

HMAC ist, wie der Name sagt, ein Message Authentication Code (MAC). Zweck eines MACs ist es die Authentizität einer Nachricht sicherzustellen. Ein Angreifer darf also nicht in der Lage sein einen gültigen MAC für eine von ihm erstellte Nachricht zu erstellen. Im Kontext von TOTP möchte der Angreifer also den MAC für die Nachricht C, dem streng monoton steigenden Zähler, erstellen. Das HMAC-Verfahren gilt derzeit für die Verwendung als MAC als sicher und wird unter anderem in TLS verwendet, um verschlüsselte Internetverbindungen zu authentifizieren. Es kann also davon ausgegangen werden, dass die erste geforderte Eigenschaft erfüllt ist.

Die in TOTP verwendete HMAC ist HMAC-SHA1, das Verfahren basiert also auf dem Secure Hash Algorithm 1. Diese Hash-Funktion erfüllt das Strict Avalance Criterion [MI16], da sich jeder kleinsten Änderung an der Eingabe jedes Bit der Ausgabe mit einer Wahrscheinlichkeit von 50% ändert. Die Ausgabe des Secure Hash Algorithm (SHA) 1 ist also zufällig verteilt.

Beide Kriterien werden also von der verwendeten HMAC erfüllt. Für einen Angreifer ist es folglich schwierig die Ausgabe der HMAC vorherzusehen. Es bleibt zu untersuchen, ob die späteren Operationen die Eigenschaften verletzen:

Durch die Konstruktion des Wertes O (Algorithmus 2.1) wird erreicht, dass jedes Byte der Ausgabe gleich wahrscheinlich in die Berechnung des Einmalkennwortes eingeht. Ungenutzt bleiben lediglich die oberen 4 Bit des letzten Bytes. Bei der Berechnung des Modulos entsteht jedoch ein leichtes statistisches Ungleichgewicht. Die Zahl S ist ein 31-Bit-Integer, also eine Zahl zwischen 0 und 2.147.483.647. Bei der Berechnung des Modulos mit dem Wert 10^6 sind die Zahlen von 000.000 bis 483.647 somit wahrscheinlicher als die Zahlen von 483.648 bis 999.999. Die Differenz liegt dabei jedoch unter 0,0001%. Um das Ungleichgewicht zu korrigieren, müsste der Algorithmus jedoch verkompliziert werden, da im Falle einer Zahl größer als 2.146.999.999 ein alternativer Wert gefunden werden müsste. Dabei muss sichergestellt werden, dass dieser Wert gleichverteilt wäre, da andernfalls keine Verbesserung eintreten würde. Den Entwicklern war das Ungleichgewicht der Werte bekannt, sie entschieden sich aufgrund der geringen Differenz der Wahrscheinlichkeit jedoch dazu den Algorithmus so zu belassen, wie er ist, da dieser Umstand das Verfahrens nicht nennenswert schwächt [RFC4226, Appendix A].

Bei dem TOTP-Verfahren ist es also auch einem versierten Angreifer nicht möglich, ohne Kenntnis des Shared Secrets ein gültiges Einmalkennwort zu erzeugen. Dies kann in einer konkreten technischen Umsetzung jedoch anders aussehen: Durch einen Seitenkanalangriff kann es möglich sein Informationen über das aktuell gültige Einmalkennwort zu erhalten. Beispielsweise durch einen Timing-Angriff, wenn bei der Überprüfung des Einmalkennworts der Vergleich abgebrochen wird, sobald eine Differenz festgestellt wurde.

Ebenso ist das TOTP-Verfahren, genauso wie reguläre Kennwörter, anfällig für einen Phishing-Angriff. Eine Phishing-Seite könnte den Benutzer nach der Eingabe seines regulären Kennworts ebenfalls nach dem aktuell gültigen Einmalkennwort fragen und dieses anschließend zum Login nutzen. In dieser Hinsicht bietet der zweite Faktor in Form von TOTP kein nennenswertes Mehr an Sicherheit, wenn die Angreifer die entsprechende Funktion in ihrer Phishing-Seite implementieren. Wenn sie dies nicht tun, weil der Großteil der Nutzer keine Mehrfaktorauthentifizierung nutzt, dann bietet TOTP einen zusätzlichen Schutz.

2.3.4. Vergleich mit anderen Einmalkennwort-Verfahren

Ähnlich wie sich Authentifizierungsverfahren in die drei eingangs genannten Kategorien (Abschnitt 2.1) einordnen lassen, lassen sich auch die Einmalkennwort-Verfahren kategorisieren:

Zeitbasiert TOTP

Zählerbasiert HOTP, Lamport One-Time Password (OTP)

Getrennter Kanal TAN-Liste, SMS

Challenge-Response Zero-Knowlege-Beweis

Gegenüber zählerbasierten Verfahren bieten zeitbasierte Verfahren den Vorteil, dass der Zustand der Berechnung des Einmalkennworts bei Anbieter und Nutzer identisch ist. Es kann bei einem zählerbasierten Verfahren leicht passieren, dass der Nutzer versehentlich ein neues Einmalkennwort erzeugen lässt und anschließend nicht nutzt, weil er sich gar nicht authentifizieren wollte. Da der Anbieter keine Kenntnis darüber besitzt, dass ein Einmalkennwort ungenutzt blieb, erwartet dieser beim nächsten Authentifizierungsversuch das versehentlich generierte Kennwort anstatt des Kennworts, das dem Nutzer angezeigt wird. Dies hat zur Folge, dass der Nutzer sich nicht erfolgreich authentifizieren kann, obwohl er seiner Meinung nach das korrekte Kennwort übermittelt hat. Im Gegenzug ist ein zählerbasierter Generator in der Herstellung günstiger, da dieser keine integrierte Uhr besitzen muss. Im Zeitalter der Smartphones mit kostenfreien TOTP-Generatoren ist dieser Vorteil jedoch eher theoretischer Natur.

Die Übermittlung von Einmalkennwörtern über einen getrennten Kanal in Form einer TAN-Liste bietet den Vorteil, dass sie komplett ohne weitere Technik auskommt. Dieser Vorteil ist jedoch auch der größte Nachteil: Die Anzahl der Einmalkennwörter ist begrenzt und somit muss die Liste regelmäßig erneuert werden. Dies verursacht laufende Kosten in Form von Porto und ist möglicherweise mit langen Wartezeiten verbunden. Selbst wenn die Liste in digitaler Form zur Verfügung gestellt werden würde, ist es erforderlich regelmäßig an die Erneuerung zu denken. Im Ernstfall steht möglicherweise kein gültiges Einmalkennwort zur Verfügung. Andere Verfahren, wie TOTP, bieten den Vorteil, dass diese Einmalkennwörter in unbegrenzter Menge zur Verfügung stellen

können. Die Übermittlung in Form von SMS teilt den Nachteil der Begrenztheit der TAN-Liste nicht, erfordert aber beim Anbieter spezielle Hardware zur Kommunikation mit dem Mobilfunknetz und verursacht laufende Kosten. Auch ist es möglich, dass die Nachricht erst mit erheblicher Verzögerung beim Nutzer ankommt, beispielsweise in Gegenden mit schlechter Netzabdeckung oder in Gebäuden. TOTP funktioniert komplett ohne Kommunikation mit der Außenwelt.

Der Übergang zwischen einem Challenge-Response-Einmalkennwortverfahren und Smartcard-basierenden Verfahren ist fließend, ein ausreichend sicheres Verfahren ist kaum im Kopf zu lösen und wenn die Challenge manuell in ein Programm übertragen werden muss, dann ist der Schritt bis zur Smartcard nicht mehr weit. Wenn nicht mit einer Smartcard gearbeitet wird, dann haben Challenge-Response-Verfahren den Nachteil, dass ein hoher Aufwand zur Authentifizierung erforderlich ist; wenn mit einer Smartcard gearbeitet wird, dann treffen die in Abschnitt 2.2.4 beschriebenen Vor- und Nachteile zu.

Fazit

Es wird deutlich, dass das TOTP-Verfahren im Bereich der Benutzerfreundlichkeit viele Vorteile gegenüber alternativer Einmalkennwortverfahren bietet. Durch die Standardisierung lassen sich mittels einer Client-Anwendung eine Vielzahl von Diensten sichern, die Kennwörter sind durch die geringe Länge einfach zu übertragen und die erstmalige Einrichtung ist durch die Unterstützung für QR-Codes ohne besondere Kenntnisse zu bewerkstelligen. TOTP ist auch in Bezug auf die technischen Implementierung attraktiv, so ist der Algorithmus zur Generierung der Einmalkennwörter sehr simpel aufgebaut (siehe Algorithmus 2.1) und lässt wenig Raum für sicherheitsrelevante Programmierfehler.

2.4. Verlust des zweiten Faktors

Es wird unweigerlich passieren, dass ein Nutzer des Dienstes die Möglichkeit verliert sich mit dem gewählten zweiten Faktor zu authentifizieren. Ein Smartphone mit der TOTP-Client-Anwendung könnte beispielsweise zu Boden fallen oder gestohlen werden und dem Nutzer dadurch nicht mehr zur Verfügung stehen. Oder aber der Nutzer deinstalliert die Client-Anwendung, um Spei-

cherplatz zu schaffen, und vergisst vorher die Zweifaktor-Authentifizierung bei den betroffenen Diensten zu deaktivieren. Es sollte daher ein Konzept entwickelt werden, wie man dem Benutzer in diesem Fall helfen kann wieder an sein Benutzerkonto zu gelangen, ohne die erhöhte Sicherheit durch die Zweifaktor-Authentifizierung zu untergraben. Ein einfaches Senden eines Links an die hinterlegte E-Mail-Adresse wäre nicht ausreichend, da ein Angreifer mit Kontrolle über das E-Mail-Konto des Nutzers auf diese Weise sowohl das Kennwort zurücksetzen, als auch die Zweifaktor-Authentifizierung deaktivieren könnte.

Wenn dem Anbieter des Dienstes die Adressdaten des Nutzers bekannt sind (beispielsweise in einem Online-Shop), dann könnte er den Nutzer beispielsweise durch einen unterschriebenen Brief zusammen mit einer Kopie des Personalausweises authentifizieren. Eine andere Möglichkeit ist es, mehrere sichere Verfahren zur Zweifaktor-Authentifizierung parallel anzubieten. Wenn einer der Faktoren unbrauchbar wird, dann hat der Nutzer die Möglichkeit auf eine Alternative auszuweichen. So ist es üblich, bei der ersten Einrichtung eine Liste von Einmalkennwörtern, ähnlich einer TAN-Liste, zu generieren, die der Benutzer auf unterschiedlichste Art und Weise aufbewahren kann.

3. Implementation von Universal Second Factor

Nachdem in Abschnitt 2.2 bereits untersucht wurde, wie Universal Second Factor (U2F) laut Spezifikation implementiert werden sollte, soll sich Kapitel 3 der konkreten Umsetzung einer Authentifizierung mittels U2F widmen. Dazu wird zuerst einmal die bestehende Anwendung vorgestellt (Abschnitt 3.1), anschließend die notwendige technische Basis für die Mehrfaktor-Authentifizierung geschaffen (Abschnitt 3.2). Nachdem alles vorbereitet wurde, wird untersucht, wie die Kommunikation mit der Smartcard abläuft und die in Abschnitt 2.2.2 kennengelernten „Responses“ verarbeitet werden (Abschnitt 3.3), zweitens wird schließlich demonstriert, wie das Verfahren in das System eingebettet wird (Abschnitte 3.4, 3.5).

Ergebnis der Implementierung sollen zwei Dinge sein:

1. Eine wiederverwendbare U2F-Bibliothek, die die Kommunikation mit der Smartcard übernimmt.

2. Eine flexible Integration einer Mehrfaktor-Authentifizierung, sodass ohne viel Aufwand weitere Verfahren (wie beispielsweise TOTP) ergänzt werden können.

3.1. Vorstellung des bestehenden Systems

Die bestehende Anwendung fitnessKOMPLEX ist in PHP 5.6 entwickelt und kommuniziert via FastCGI mit dem Webserver (Apache 2). Das Betriebssystem der Server ist Red Hat Enterprise Linux (RHEL) 5. Der fitnessKOMPLEX speichert seine Daten in einer SQL-Datenbank.

Die bestehende Authentifizierung ist direkt im Webserver implementiert und kann über verschiedene Verfahren erfolgen. Im Regelfall wird der Nutzer über

das HTTP-Digest-Verfahren authentifiziert [RFC2617, Abschnitt 3]. Ein alternatives Verfahren ist die Authentifizierung über ein TLS-Clientzertifikat.

Allen Verfahren ist gemein, dass der Webserver den authentifizierten Nutzernamen als Umgebungsvariable an die PHP-Anwendung weitergibt. Die Anwendung muss sich um nichts kümmern[1]. Das Kennwort des Nutzers wird zu keinem Zeitpunkt an den fitnessKOMPLEX weitergegeben. Aufgrund dieser Tatsache existiert in der Anwendung noch kein Sitzungssystem. Alle Anfragen erfolgen zustandslos und der Benutzer wird bei jedem Seitenabruf erneut vom Webserver authentifiziert.

3.2. Voraussetzungen schaffen

3.2.1. Sitzungssystem

Für das U2F-Verfahren und die Mehrfaktor-Authentifizierung im Allgemeinen ist es erforderlich, Sitzungen verwalten zu können: Es muss möglich sein festzustellen, ob der Nutzer die Seite zum ersten Mal öffnet und daher nach dem zweiten Faktor gefragt werden sollte, oder, ob er sich bereits erfolgreich authentifiziert hat. Für das U2F-Verfahren im Speziellen müssen die an die Smartcard gesendeten Challenges gespeichert werden, damit die Responses anschließend überprüft werden können.

PHP integriert bereits ein Sitzungssystem in der Standardbibliothek [The]. Dieses ist allerdings nicht nutzbar, da die Webserver als Cluster betrieben und die Sitzungen nicht zwischen den einzelnen Knoten synchronisiert werden. Je nachdem, ob man bei einem späteren Aufruf auf den gleichen Knoten gelangt, besteht die Möglichkeit, dass die Sitzung als ungültig erkannt wird. Dies ist offensichtlich unbefriedigend. Die naheliegende Alternative wäre es, die Sitzungen in der bestehenden SQL-Datenbank zu speichern. Dies würde jedoch bedeuten, dass ebenfalls ein Verfahren zum Aufräumen abgelaufener Sitzungen gefunden werden müsste. Andernfalls würde die Datenbank irgendwann mit vielen Sitzungen unnötig aufgebläht. Für zuverlässige Aufräumarbeiten wären zeitgesteuerte Aufgaben (Cronjobs) erforderlich, die aber nicht zur Ver-

[1] Tatsächlich wird die Anwendung nicht einmal gestartet, wenn für den aufgerufenen URL die Authentifizierung aktiviert ist und der Nutzer nicht authentifiziert werden kann. Die Entscheidung, ob eine Authentifizierung erfolgen soll oder nicht, wird auf Basis der verwendeten Subdomain getroffen.

fügung stehen. Stattdessen setzt die Beispielimplementierung zur Verwaltung der Sitzungen auf ein verschlüsseltes, authentifiziertes Cookie [RFC6265]. Dies hat den Vorteil, dass der Verantwortungsbereich für die Sitzungen nicht beim Webserver, sondern beim Webbrowser liegt. Dieser hat dafür Sorge zu tragen, dass die Sitzungsdaten zuverlässig gespeichert und der Anwendung zur Verfügung gestellt werden. Die Anwendung selbst bleibt in dieser Hinsicht zustandslos. Ein etwaiges Ablaufdatum der Sitzung wird dabei innerhalb der verschlüsselten Daten abgelegt. Das Datum, das beim Setzen des Cookies mitgesendet wird, könnte vom Nutzer manipuliert werden.

Zur Sicherung des Cookies wird auf die php-encryption-Bibliothek in Version 1.2.1 gesetzt [Hor]. Eine neuere Version der Bibliothek ist aufgrund der in RHEL 5 mitgelieferten FIPS-zertifizierten OpenSSL-Version inkompatibel [Arc16]. Die Verwendung einer Bibliothek empfiehlt sich, da diese besser getestet ist als eine manuell (auf Basis von OpenSSL) implementierte Verschlüsselungsroutine es jemals sein wird.

3.2.2. Datenbanktabellen

Die Informationen zur Mehrfaktor-Authentifizierung sollen ebenfalls in der SQL-Datenbank hinterlegt werden. Dazu sind zwei Änderungen am Datenbank-Schema erforderlich:

1. Eine neue boolesche Spalte in der Nutzer-Tabelle (`Person`), welche spezifiziert, ob für dieses Nutzerkonto eine Mehrfaktor-Authentifizierung erfolgen soll.

2. Eine neue Tabelle (`2fa`), in der die eingerichteten Authentifizierungsmerkmale hinterlegt werden.

Die einzelnen Spalten der `2fa`-Tabelle (Abbildung 3.1) sind:

id Die `id` des Nutzers zu dem dieses Merkmal gehört.

type Die Art des Merkmals. In der Beispielimplementierung entweder `u2f` oder `scratch`.

device Ein eindeutiger, vom Benutzer gewählter Name für das Merkmal (beispielsweise „schwarze Smartcard“).

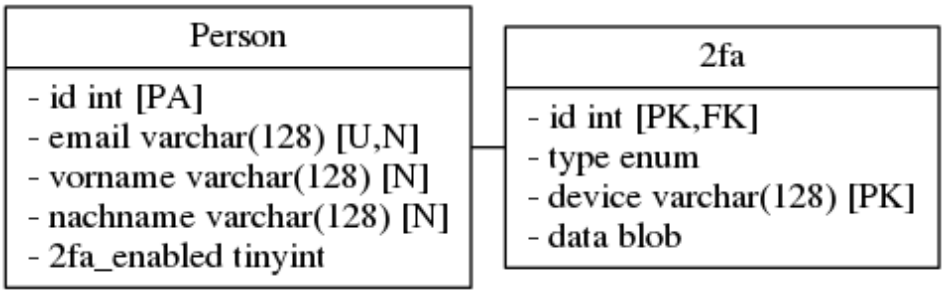


Abbildung 3.1.: Datenbankschema mit den notwendigen Änderungen für Mehrfaktor-Authentifizierung.

data Ein opaker Bytestring mit Metadaten des Authentifizierungsmerkmals. Der Inhalt ist abhängig von `type` zu interpretieren (beispielsweise der öffentliche Schlüssel, Abschnitt 2.2.2).

3.2.3. OpenSSL

Voraussetzung für U2F ist eine Kryptografie-Bibliothek mit Unterstützung für ECDSA-Signaturen auf der P-256-Kurve (Abschnitt 2.2.1). In der in RHEL 5 mitgelieferten OpenSSL-Version fehlt neben der Unterstützung für den Advanced Encryption Standard (AES) im CTR-Modus (Abschnitt 3.2.1) auch die Unterstützung für Kryptografie auf Basis von elliptischen Kurven. Anders als bei der php-encryption-Bibliothek, besteht bei der Kommunikation mit der Smartcard aber die Möglichkeit diese Unzulänglichkeit sinnvoll zu umgehen. Anpassungen an einer Fremdbibliothek würden die Vorteile des Einsatzes selbiger ad absurdum führen. Bei der selbst entwickelten Bibliothek zur Kommunikation mit der Smartcard kann die Problematik hingegen direkt bei der Entwicklung berücksichtigt werden.

Anstatt auf die in PHP integrierte Funktion zum Validieren von Signaturen zu setzen (`openssl_verify`), soll alternativ ein Kommandozeilenaufruf einer geeigneten OpenSSL-Version verwendet werden können (`openssl dgst`). Dazu wird eine statisch kompilierte OpenSSL-Binary in der aktuellen (1.0.1c) Version mitgeliefert. Diese kann dann über die `exec`-Funktion mittels PHP angesprochen werden. Die Verwendung einer statisch kompilierten Binary bietet hier den Vorteil, dass alle Abhängigkeiten von OpenSSL in der Binary enthalten sind. Diese Binary kann also auf jedem Rechner verwendet werden,

auf dem ein Linux-Kernel (Version 2.6 oder höher) läuft - unabhängig davon, in welcher Version die Systembibliotheken vorliegen.

3.3. Kommunikation mit der Smartcard

Zur Kommunikation mit der Smartcard wurde eine wiederverwendbare PHP-Klasse entwickelt. Das öffentliche API dieser Klasse besteht, neben dem Konstruktor, aus 4 Methoden: Jeweils eine Methode zum Generieren der Challenge und Verarbeiten der Response für beide der Funktionen, die die Smartcard zur Verfügung stellt:

generateRegisterRequest Berechnet die notwendigen Eingabeparameter für einen `u2f_register_request`.

verifyRegisterResponse Validiert und verarbeitet die Antwort auf einen `u2f_register_request`.

generateSignRequest Berechnet die notwendigen Eingabeparameter für einen `u2f_sign_request`.

verifySignResponse Validiert und verarbeitet die Antwort auf einen `u2f_register_request`.

Die einzelnen Methoden sind so entwickelt worden, dass es möglichst schwierig ist das API fehlerhaft zu verwenden und dadurch versehentlich Sicherheitslücken einzuführen. So werden von den `verify*Response`-Methoden Exceptions geworfen, wenn die Antwort der Smartcard nicht gültig validiert werden konnte. Dadurch wird sichergestellt, dass es, anders als beim Rückgabewert, nicht möglich ist zu vergessen zu überprüfen, ob die Antwort gültig war. Wenn die Exception nicht behandelt wird, dann wird die Anfrage automatisch seitens der Laufzeitumgebung abgebrochen. Ebenso wurden die Parameter der Methoden so gewählt, dass die Antwort der Smartcard unverändert genutzt werden kann. Dadurch muss der Nutzer der Klasse die Antwort lediglich übergeben, ohne vorab komplexe Vorarbeit leisten zu müssen.

Auf Seiten des Webbrowsers setzt die Integration auf eine von Google entwickelte JavaScript-Bibliothek, die das von der FIDO Alliance spezifizierte High-Level-JavaScript-API zur Verfügung stellt [BBL15; Goo14].

Alle nachfolgend kennengelernten Strukturen werden in der Kommunikation mittels JavaScript Object Notation (JSON) kodiert.

3.3.1. `__construct`

Die einzige Aufgabe des Konstruktors ist es, zwei Parameter, die für mehrere Methoden benötigt werden, zu hinterlegen: Die `appId` (Abschnitt 2.2.2) und ein Pfad zu einer OpenSSL-Binary, falls die in PHP integrierte Version unzulänglich ist (siehe Abschnitt 3.2.3).

3.3.2. `generateRegisterRequest`

`generateRegisterRequest` berechnet den notwendigen Inhalt für die `RegisterRequest`-Struktur, welche in der `u2f.sign`-Methode des JavaScript-API ein notwendiger Parameter ist.

```
1 dictionary RegisterRequest {
2         DOMString version;
3         DOMString challenge;
4 };
```

Listing 3.1: Aufbau der RegisterRequest-Struktur. [BBL15, Abschnitt 5.1]

`version` wird fest als `U2F_V2` gewählt. `challenge` ist ein mit einem kryptografisch sicheren Zufallszahlengenerator erzeugter Bytestring der Länge 32, der mittels websafe-base64 kodiert wurde [RFC4648, Abschnitt 5].

3.3.3. `verifyRegisterResponse`

`verifyRegisterResponse` ist als Gegenstück zu `generateRegisterRequest` dafür zuständig, die `RegisterResponse`-Struktur in der Antwort der Smartcard zu verarbeiten.

```
1 dictionary RegisterResponse {
2         DOMString version;
3         DOMString registrationData;
4         DOMString clientData;
5 };
```

Listing 3.2: Aufbau der RegisterResponse-Struktur [BBL15, Abschnitt 5.1.3].

`registrationData` ist die, in websafe-base64 kodierte, Antwort auf die Anfrage (siehe Abbildung 2.2). `clientData` ist eine in websafe-base64 kodierte Struktur des Typs `ClientData`.

```
1 dictionary ClientData {
2         DOMString typ;
3         DOMString challenge;
4         DOMString origin;
5         (DOMString or JwkKey) cid_pubkey;
6 };
```

Listing 3.3: Aufbau der ClientData-Struktur [BE15, Abschnitt 7].

Aufgabe der Methode ist es, die einzelnen Bestandteile der `RegisterResponse`-Struktur zu verarbeiten und zu prüfen. Die Prüfung von `clientData` ist trivial:

1. `typ` muss `navigator.id.finishEnrollment` sein.
2. `challenge` muss dem Wert `challenge` des zugehörigen `RegisterRequest` entsprechen (Abschnitt 3.3.2).
3. `origin` wird, wenn gewünscht, vom aufrufenden Code überprüft. Der Wert steht in keiner direkten Beziehung zum U2F-Protokoll selbst, eine fehlende Prüfung verringert die Sicherheit des Verfahrens nicht. Die Prüfung des Werts dient dem Schutz vor Phishing-Angriffen.
4. `cid_pubkey` wird, wie `origin`, vom aufrufenden Code überprüft. Damit dieses Feld überhaupt befüllt wird, ist eine spezielle TLS-Konfiguration innerhalb des Webservers erforderlich.

Die Prüfung von `registrationData` gestaltet sich ein wenig komplexer:

1. Extrahieren der ersten 3 Felder über feste Byte-Offsets und Länge.
2. Extrahieren des 4. Feldes. Die Länge wird durch das 3. Feld spezifiziert. Das 3. Feld ist dazu als 8-Bit-Zahl zu interpretieren.
3. Extrahieren des 5. Feldes (Attestierungszertifikat). Die Länge wird durch das eindeutige Parsen des X.509-Zertifikats bestimmt (siehe unten).
4. Extrahieren des letzten Feldes (Signatur). Das Feld erstreckt sich bis zum Ende des Bytestrings.

5. Das 1. Feld muss den Wert `0x05` besitzen.

6. Alle Felder fester Länge müssen diese Länge besitzen.

7. Die Signatur muss durch das Attestierungszertifikat gültig signiert worden sein. Der Aufbau der von der Signatur authentifizierten Daten ist in Abbildung 2.2 abzulesen.

Das Attestierungszertifikat ist DER-kodiert in dem Bytestring eingebettet. Um es zu extrahieren, ist es erforderlich die Länge der äußersten `SEQUENCE` der `Certificate` ASN.1-Struktur zu ermitteln. Wie genau dies zu tun ist, ist in X.680 und X.690 spezifiziert [X.680, Abschnitt 8.4] [X.690, Abschnitte 8.1.2, 8.1.3.5, 8.9.1, 10.1].

Zur Prüfung der Signatur mittels OpenSSL ist es erforderlich das Zertifikat in das Privacy Enhanced Mail (PEM)-Format zu überführen. Dazu ist es lediglich erforderlich das Zertifikat im DER-Format mittels base64 zu kodieren und in das PEM-Armoring einzubetten:

```
1 -----BEGIN CERTIFICATE-----
2 ...
3 -----END CERTIFICATE-----
```

Listing 3.4: PEM-Armoring von Zertifikaten.

Nachdem die Antwort überprüft wurde, gibt die Methode eine opake Struktur zur Verarbeitung mittels `generateSignRequest` und `verifySignResponse` zurück. Diese Struktur enthält den von der Smartcard übermittelten öffentlichen Schlüssel, Schlüsselbezeichner und die U2F-Version. Außerdem wird ein Standardwert für den streng monoton steigenden `counter` (0) eingebettet.

3.3.4. `generateSignRequest`

Diese Methode arbeitet ganz analog zu `generateRegisterRequest`. Ein Unterschied besteht darin, dass sie ein Array von Schlüsseln entgegennimmt, die bereits mit dem Benutzerkonto verknüpft sind. Ein Schlüssel wird in diesem Array durch die Struktur, die von `verifyRegisterResponse` zurückgegeben wird, repräsentiert. Ein weiterer Unterschied ist, dass der Rückgabewert keine vordefinierte Struktur ist, sondern alle Felder der Struktur als einzelne Parameter an das JavaScript-API übergeben werden.

3.3.5. `verifySignResponse`

Analog zu `verifyRegisterResponse` verarbeitet diese Methode die Antwort auf eine Authentifizierungsanfrage, wie sie mit `generateSignRequest` erzeugt wurde.

```
1 dictionary SignResponse {
2         DOMString keyHandle;
3         DOMString signatureData;
4         DOMString clientData;
5 };
```

Listing 3.5: Aufbau der SignResponse-Struktur [BBL15, Abschnitt 5.2.2].

`keyHandle` ist der Bezeichner des Schlüssels, der schlussendlich die Authentifizierungsanfrage bestätigt hat. `signatureData` ist die, in websafe-base64 kodierte, Antwort auf die Anfrage (siehe Abbildung 2.3). `clientData` ist, wie bei `verifyRegisterResponse`, eine websafe-base64-kodierte `ClientData`-Struktur.

Die Prüfung von `clientData` erfolgt mit dem einzigen Unterschied, dass der `typ` auf `navigator.id.getAssertion` lauten muss, identisch zu der Prüfung in `verifyRegisterResponse`.

`signatureData` ist wie folgt zu prüfen:

1. Extrahieren aller 3 Felder über feste Byte-Offsets und Länge.
2. Überprüfung der Bits des 1. Feldes (das Byte muss die Wertigkeit `0x01` besitzen).
3. Ermitteln des verwendeten Schlüssels auf Basis von `keyHandle`.
4. Die Signatur muss durch den verwendeten Schlüssel gültig signiert worden sein. Der Aufbau der von der Signatur authentifizierten Daten ist in Abbildung 2.3 abzulesen.
5. Der `counter`-Wert muss höher sein als der zuletzt bekannte `counter`-Wert.

Damit der bei der Registrierung erhaltene öffentliche Schlüssel zur Signaturprüfung genutzt werden kann, ist es erforderlich ihn zu vervollständigen (Abschnitt 2.2.2). In der Implementierung erfolgt dies bereits in `verifyRegister`

`Response`, da es nicht notwendig ist die Vervollständigung bei jeder Authentifizierung von vorne vorzunehmen. Zur Vervollständigung wird die DER-Kodierung der ASN.1-Struktur eines öffentlichen Schlüssels aufgebaut und der erhaltene Kurvenpunkt darin eingebettet. Wie genau dies abläuft, ist in RFC 5480, X.680 und X.690 definiert [RFC5480, Abschnitt 2] [X.680, Abschnitt 8.4] [X.690, Abschnitte 8.1.2, 8.6.2.2, 8.19.1, 10.2]. Analog zum Attestierungszertifikat in `verifyRegisterResponse` wird der Schlüssel anschließend in das PEM-Format konvertiert und dann an OpenSSL zur Signaturprüfung übergeben.

```
1 -----BEGIN PUBLIC KEY-----
2 ...
3 -----END PUBLIC KEY-----
```

Listing 3.6: PEM-Armoring von öffentlichen Schlüsseln.

Wenn die Antwort gültig war, wird der `counter`-Wert des verwendeten Schlüssels aktualisiert und das aktualisierte Array der übergebenen Schlüssel zurückgegeben, damit diese in der Datenbank hinterlegt werden können.

3.4. Aktivierung des zweiten Faktors

Wenn der Nutzer eine neue Smartcard mit seinem Konto verknüpfen möchte, dann sind dazu zwei Dinge erforderlich:

1. Der eindeutige Name für die Smartcard.
2. Der öffentliche Schlüssel der Smartcard.

Ersteres wird durch ein simples Formularfeld erfragt. Für letzteres wird beim Aufruf des Formulars mittels `generateRegisterRequest` eine Challenge erstellt und in der Sitzung hinterlegt. Die notwendigen Daten für das JavaScript-API werden direkt in den Hypertext Markup Language (HTML)-Code der Einrichtungsseite eingebettet. Durch Betätigen eines Knopfes sendet der Nutzer die Challenge an die Smartcard und nachdem er die Nutzung der Smartcard autorisiert, sendet diese die Antwort zurück an den Webbrowser. Dieser ruft dann die an das API übergebene Callback-Funktion auf und diese hinterlegt die Antwort in einem versteckten Formularfeld. Wenn der Benutzer mit seiner Eingabe zufrieden ist, sendet er das Formular ab.

Der Code, welcher das Formular entgegennimmt, prüft, ob alle Felder ausgefüllt wurden und ob eine gültige Challenge in der Sitzung hinterlegt ist. Wenn dies der Fall ist, wird die Antwort der Smartcard an `verifyRegisterResponse` übergeben und im Erfolgsfall wird ein neuer Datensatz für die Smartcard in der Datenbank hinterlegt. Der Benutzer wird dann zurück in die Liste seiner Authentifizierungsmerkmale geleitet.

3.5. Überprüfung des zweiten Faktors

Sobald die Mehrfaktor-Authentifizierung für ein Konto aktiviert ist, muss der zweite Faktor in jeder neuen Sitzung überprüft werden. Dazu wird in der Sitzung der Zeitpunkt hinterlegt, zu dem zuletzt eine Mehrfaktor-Authentifizierung durchgeführt wurde. Wenn dieser Zeitpunkt mehr als zwei Stunden[2] verstrichen ist (oder bislang kein Zeitpunkt hinterlegt ist), dann ist es erforderlich eine neue Authentifizierung durchzuführen. Um diese Überprüfung sicherzustellen, wird unmittelbar nach Ermittlung des Nutzers und Lesen der Sitzung überprüft, ob mehr als zwei Stunden verstrichen sind. Wenn dies der Fall ist und der aufgerufene URL nicht explizit freigeschaltet wurde (beispielsweise Impressum und die Seiten, die die Authentifizierung durchführen), dann wird der Nutzer in eine Liste seiner hinterlegten Merkmale umgeleitet und die Anfrage abgebrochen. In dieser Liste wählt der Nutzer dann das Merkmal, das er verwenden möchte, und gelangt danach auf ein Formular mit dem er die Authentifizierung durchführen kann.

Im Falle von U2F wird analog zur Aktivierung des zweiten Faktors die Challenge generiert und in der Sitzung hinterlegt. Ebenso enthält der HTML-Code die notwendigen Parameter für das JavaScript-API. Mit Betätigen eines Knopfes sendet der Nutzer die Authentifizierungsanfrage an seine Smartcard. Nachdem er die Nutzung der Smartcard autorisiert hat, sendet diese die Antwort zurück an den Webbrowser, der die Callback-Funktion aufruft. Diese hinterlegt die Antwort in einem versteckten Formularfeld und sendet das Formular selbstständig ab.

Die Überprüfung der gesendeten Formulardaten erfolgt analog zur Aktivierung mittels `verifySignResponse`. Wenn die Daten gültig sind, wird in der

[2] Dieser Zeitraum sollte der durchschnittlichen Sitzungslänge entsprechend gewählt werden. Wenn der Großteil der Nutzer die Anwendung mehr als zwei Stunden am Stück nutzt, dann sollte die Gültigkeitsdauer der Authentifizierung entsprechend erhöht werden.

Sitzung der Zeitpunkt der letzten Authentifizierung hinterlegt und die Metadaten der Smartcard in der Datenbank aktualisiert, damit der `counter`-Wert aktuell ist. Anschließend wird der Nutzer auf die Startseite der Anwendung zurückgeleitet und kann sie, wie gewohnt, verwenden.

3.6. Fazit

Wie in den Abschnitten 3.2.2, 3.4 und 3.5 deutlich wird, sind nur geringe Änderungen am Bestandssystem erforderlich, um es mit einer Mehrfaktor-Authentifizierung auszustatten. Neben den zwei Änderungen an der Datenbank muss lediglich eine Überprüfung, ob der Nutzer bereits die Mehrfaktor-Authentifizierung durchgeführt hat, direkt nach der Initialisierung der Sitzung ergänzt werden. Alle anderen Änderungen waren entweder vorbereitende Maßnahmen, die in vielen Systemen bereits umgesetzt sind, oder neue Formulare, welche die Einrichtung und Prüfung der Merkmale vornehmen. Es ist also leicht möglich zu prüfen, dass die Änderungen am Bestandscode der Anwendung korrekt sind und keine sicherheitskritischen Fehler einführen.

Zur Prüfung der zweiten Zielsetzung wurde in der begleitenden Implementierung neben der Unterstützung für U2F eine Unterstützung für Notfallcodes (ähnlich einer TAN-Liste) umgesetzt. Dafür war es lediglich erforderlich die erlaubten Werte der `type`-Spalte in der `2fa`-Tabelle zu erweitern und zwei neue Formulare zur Einrichtung und Prüfung der Notfallcodes zu ergänzen. Am PHP-Quelltext der Anwendung waren keine *Änderungen* erforderlich. Insgesamt wurden nur rund 220 Zeilen Code, die dadurch leicht von einem zweiten Entwickler überprüft werden können, *ergänzt*. Das System ist also modular erweiterbar, falls in Zukunft weitere Verfahren gewünscht werden.

4. Autorisierung

Die Realisierung einer sicheren Authentifizierung mittels mehrerer Faktoren stand im Fokus von Kapitel 2 und 3. Damit ist die Zugriffskontrolle jedoch nicht abgeschlossen. Nachdem jetzt sicher bestimmt werden kann, *wer* die Anwendung gerade verwendet (Authentifizierung), muss auch noch überprüft werden, *was* diejenige Person in der Anwendung einsehen und verwenden darf (Autorisierung). Eine sichere Autorisierung ist ohne eine sichere Authentifizierung nicht zu realisieren: Es ist nicht zielführend bestimmte Funktionen nur für den Geschäftsführer eines Unternehmens zur Verfügung zu stellen, wenn sich gegenüber der Anwendung jeder als Geschäftsführer ausgeben kann und dies von der Anwendung akzeptiert wird. Umgekehrt ist die Authentifizierung ohne gesonderte Autorisierung in vielen - aber nicht allen - Fällen ebenso unbefriedigend, wie in Abschnitt 4.1 noch genauer diskutiert wird.

Dieses Kapitel soll sich daher mit der Realisierung einer geeigneten Autorisierungsstrategie beschäftigen. Dabei werden verschiedene Konzepte zur Umsetzung einer Autorisierung vorgestellt, die jeweils Vor- und Nachteile in Bezug auf technische Komplexität, Mächtigkeit und administrative Verständlichkeit haben.

Für einige Anwendungsfälle kann es sinnvoll sein, dass Zugriffsrechte dynamisch im System hinterlegt werden können. Beispielsweise könnte ein Profil in einem sozialen Netzwerk nur für ausgewählte Freunde zugänglich gemacht werden sollen. Dazu könnte bei der Registrierung eines Nutzers automatisch ein Recht „Kann Profil von Max Mustermann aufrufen“ erzeugt und bei der Löschung des Kontos wieder entfernt werden. Derartige Rechte sollten dem Administrator in der zentralen Verwaltung, auch der Übersichlichkeit halber, nicht angezeigt werden. Es wäre aber denkbar für Teammitglieder ein zentral vergebenes Zugriffsrecht „Kann alle Profile einsehen“ anzulegen, das die Prüfung des spezifischen Rechtes aushebelt.

4.1. Keine gesonderte Autorisierung

Diese Autorisierungsstrategie führt keine Autorisierung durch. Jeder authentifizierte Nutzer ist berechtigt alle Funktionen der Anwendung zu nutzen. Dieses Verfahren ist offensichtlich technisch sehr einfach umzusetzen und auch die administrative Verständlichkeit ist sehr gut: Es muss nichts programmiert und auch nichts eingestellt werden. Dafür ist das Verfahren allerdings auch nicht sehr mächtig: Nutzer dürfen entweder alles oder nichts. Aus diesem Grund eignet sich diese Art der Autorisierung nur schlecht für Anwendungen in denen der Nutzerkreis nicht überschaubar ist (etwa, weil Nutzer sich selbstständig registrieren können). Es wird unweigerlich zu Missbrauch kommen, sodass Zugriffsrechte für moderative Funktionen erforderlich werden.

Diese Autorisierungsstrategie wird beispielsweise vom Data Structure-Server Redis verwendet: Jeder, der sich zu einem Redis-Server verbinden kann, kann alle Befehle verwenden [San]. Dies bietet sich hier an, da der Kreis der Verbindungsberechtigten ohnehin sehr klein ist (die Anwendung selbst und die Systemadministratoren) und dadurch die Programmlogik von Redis einfacher gestaltet sein kann. Ein weiterer Anwendungsfall für Authentifizierung ohne Autorisierung könnte eine unternehmensinterne Informationsseite sein, die nur aus dem Intranet des Unternehmens zugänglich ist. Die Authentifizierung erfolgt über die abrufende IP-Adresse. Es wäre dann keine gesonderte Autorisierung erforderlich, wenn keine sensiblen Informationen in diese Seite eingestellt werden.

4.2. Rechtelevel

Ein Verfahren basierend auf Rechteleveln ordnet jedem Benutzer ein bestimmtes Zugriffslevel (in der Regel als Ganzzahl) zu. Den einzelnen Funktionen der Anwendung wird ebenfalls eine Zahl zugeordnet - das Mindestlevel, um diejenige Funktion nutzen zu können. In der technischen Umsetzung muss lediglich das Level des Nutzers gespeichert und bei den einzelnen Funktionen dieses Level mit einem festen Wert verglichen werden. Auf administrativer Ebene ist das Verfahren ebenfalls sehr simpel: Der Administrator muss nur das Level des Benutzers auswählen. Über welches Level welche Funktionen zur Verfügung stehen, könnte unmittelbar neben dem Eingabefeld angezeigt werden, sodass es unwahrscheinlich ist einen sicherheitsrelevanten Bedienfehler zu verursachen.

Jedoch ist auch dieses Verfahren nicht sehr mächtig: Es können nur Zugriffsrechte vergeben werden, die jeweils Teilmengen voneinander sind; ein höheres Level inkludiert auch alle Rechte eines niedrigeren Levels. Die einzelnen Kompetenzen einer komplexeren Anwendung lassen sich so im Regelfall nicht abbilden. Verglichen mit den einzelnen Rechten in einem Unternehmen sollte die PR-Abteilung keinen Zugriff auf die privaten Daten der Mitarbeiter erhalten. Umgekehrt sollte die HR-Abteilung aus Sicherheitsgründen keinen Zugriff auf die Social-Media-Konten des Unternehmens erhalten. Keine der beiden Abteilungen hat jeweils alle Rechte der anderen Abteilung, somit wären Rechtelevel nicht mächtig genug diesen Fall abzudecken.

Diese Autorisierungsstrategie wird beispielsweise in Internet Relay Chat (IRC)-Netzwerken verwendet. Pro Channel lassen sich die Level „Voice“ und „Op“ vergeben (in einigen Netzwerken noch weitere); jeder „Op“ hat dabei automatisch auch die Rechte von „Voice“. Auch die Minecraft Server-Software regelt die Rechte von Operatoren über Zugriffslevel[129].

4.3. Vergabe einzelner Rechte an Nutzer

Hierbei werden einzelne Berechtigungen flexibel direkt an die registrierten Benutzerkonten vergeben. In der technischen Umsetzung könnte dieses System durch eine n:n-Relation „Darf“ zwischen dem Benutzer und der Berechtigung realisiert werden. Zur Überprüfung der Berechtigung muss dann geprüft werden, ob die entsprechende Relation in der SQL-Datenbank existiert. Von administrativer Seite könnten die unterschiedlichen Berechtigungen im „Benutzerbearbeiten“-Formular aufgelistet und durch Checkboxen erteilt werden. Für den Administrator ist diese Autorisierungsstrategie also sehr transparent: Er sieht unmittelbar, welche Rechte ein bestimmter Nutzer besitzt und Fehlkonfigurationen sind leicht zu erkennen. Allerdings ist die Verwaltung sehr aufwändig: Wenn man einer gesamten Klasse von Nutzern (beispielsweise allen Mitarbeitern der PR-Abteilung) eine neue Berechtigung erteilen oder nehmen möchte, dann müssen eine Vielzahl von Benutzerkonten bearbeitet werden. Es kann dabei leicht passieren, dass ein Benutzer vergessen wird.

Die Mächtigkeit dieses Verfahrens ist direkt von den einzelnen Zugriffsrechten abhängig: Je feingranularer die einzelnen Rechte unterteilt werden, desto präziser lassen sich die Zugriffsrechte vergeben. Im Gegensatz zu den Rechtele-

veln (Abschnitt 4.2) können auch Strukturen umgesetzt werden, bei denen sich die einzelnen Funktionen nicht in eine Rangfolge unterteilen lassen. Die Zugriffsrechte sollten jedoch nicht zu differenziert unterteilt sein. Je mehr Rechte im System existieren, desto höher ist der Konfigurationsaufwand und die Fehleranfälligkeit.

4.4. Vergabe von Rechtegruppen an Nutzer

Rechtegruppen erweitern das in Abschnitt 4.3 vorgestellte Verfahren um eine Bündelung von Zugriffsrechten. Anstatt die Zugriffsrechte direkt an Nutzer zu verteilen, werden die Rechte zu einer Rechtegruppe zusammengefasst und diese Rechtegruppe dem Nutzer zugewiesen. Dies erleichtert es, die Zugriffsrechte für eine Vielzahl von Benutzern gleichzeitig anzupassen und umgeht die oberhalb beschriebene Problematik des Vergessens einzelner Nutzer. Mit dieser Verbesserung geht jedoch eine höhere Komplexität der technischen Umsetzung einher. Es ist eine zusätzliche Tabelle für die Gruppen erforderlich und die „Darf“-Relation besteht nicht mehr zwischen Nutzer und Berechtigung, sondern zwischen Gruppe und Berechtigung. Stattdessen muss eine neue 1:n-Relation „Mitglied von“ zwischen Nutzer und Gruppe geschaffen werden. Auf administrativer Seite werden die Einstellungen etwas intransparenter: Es ist nicht mehr direkt ersichtlich, welche einzelnen Rechte ein Nutzer hat, man muss dazu wissen, was für Rechte die jeweilige Gruppe besitzt.

Grundsätzlich ist dieses Verfahren aber gleichmächtig zu der direkten Vergabe von Rechten: Wenn man eine Rechtegruppe pro Nutzer anlegt, dann ist das Verfahren praktisch identisch zu dem aus Abschnitt 4.3. Umgekehrt lässt sich jede unterschiedliche Kombination an Rechten als Benutzergruppe darstellen und einem Nutzer zuordnen.

4.5. Vergabe mehrerer Rechtegruppen an Nutzer

Die Vergabe mehrerer Rechtegruppen an Nutzer ist eine direkte Erweiterung des vorherigen Verfahrens: Die „Mitglied von“-Relation wird zu einer n:n-Relation. Durch die Vergabe mehrerer Rechtegruppen steigt die Mächtigkeit auch dieses Verfahrens nicht an. Die Vergabe nur einer Rechtegruppe ist ein Spezialfall der Vergabe mehrerer Rechtegruppen, umgekehrt kann man für je-

de erdenkliche Kombination von Gruppen (alle Elemente der Potenzmenge) eine einzelne Gruppe anlegen, welche das Resultat der Kombination darstellt.

Genauso wie die Vergabe von Rechtegruppen die Vergabe einzelner Rechte flexibler und potentiell weniger aufwändig macht, macht die Vergabe mehrerer Rechtegruppe die Vergabe von Rechtegruppen flexibler und potentiell weniger aufwändig. Hier treffen aber die gleichen Nachteile zu: Durch die Erhöhung der Flexibilität wird das System intransparenter. Der Administrator muss nicht nur wissen, welche Rechte in einer einzelnen Gruppe vergeben sind, sondern muss die Rechte aller vergebenen Gruppen im Detail kennen.

Auf technischer Seite stellt sich bei der Vergabe mehrerer Rechtegruppen die Problematik des Zusammenführens. Ein einzelnes Recht kann in einer Vielzahl von Gruppen definiert sein. Wenn ein Recht nicht ausschließlich ein boolescher Wert ist (beispielsweise „Maximaler Speicherplatz“ in einer Cloud-Software), dann können potentiell widersprüchliche Rechte in den Gruppen definiert sein. Es muss also für jedes Recht genau definiert werden, was das Resultat ist, wenn die Rechte der Gruppen eines Nutzers einander widersprechen.

Dieses Verfahren, üblicherweise als Role-based access control (RBAC) bekannt [FK92], ist aufgrund seiner Flexibilität weit verbreitet. So wird es beispielsweise in abgewandelter Form für die Dateirechte von Unixoiden Betriebssystemen verwendet: Gruppen können drei verschiedene Rechte pro Datei vergeben werden. Benutzer können einer Vielzahl von Gruppen angehören und besitzen das Recht, wenn eine der Gruppen das Recht besitzt. Zusätzlich ist es möglich, spezifisch für den Besitzer einer Datei unterschiedliche Rechte zu definieren.

4.6. Wahl der Autorisierungsstrategie

Die in Abschnitt 4.1 und 4.2 beschriebenen Strategien sind für die in dieser Arbeit erweiterte Anwendung fitnessKOMPLEX, ein umfangreicher Fitnesstracker und -planer, unzureichend. Sportübungen sollten nur von ausgebildeten und vom Administrator explizit berechtigten Trainern angelegt werden können. Die Übungen von Laien könnten die Gesundheit der Nutzer gefährden. Trainer sind aber nicht notwendigerweise im Unternehmen angestellt und sollten daher keinen Zugriff auf die Oberfläche für die Kundenbetreuung haben. Die Kundenbetreuung wird umgekehrt aber nicht notwendigerweise eine Trai-

nerausbildung besitzen. Ebenso wäre es denkbar, bestimmte Funktionen der Anwendung nur gegen eine zusätzliche monatliche Gebühr zur Verfügung zu stellen. Wenn es mehr als eine dieser Funktionen gibt, dann sind auch hier die Zugriffsrechte keine Teilmengen voneinander.

Die anderen drei Verfahren sind, wie oberhalb festgestellt, gleichmächtig und alle in der Lage die Rechtestrukturen abzubilden. Dennoch ist nur eines der Verfahren wirklich sinnvoll: Die Pflege von Rechten, die direkt an einzelne Nutzer vergeben werden, ist mit der bereits jetzt abzusehenden Komplexität der Anwendung und den erwarteten Nutzerzahlen inpraktikabel. Die Vergabe einzelner Gruppen wäre denkbar, wenn die Anwendung ausschließlich von Nutzern mit einem klar abgegrenzten Aufgabengebiet genutzt wird. Sollten die oberhalb beschriebenen kostenpflichtigen Add-ons für die Mitgliedschaft umgesetzt werden, so wären exponentiell viele Gruppen erforderlich, um alle Kombinationen darzustellen.

Nachdem sich vier der fünf vorgestellten Verfahren für den geplanten Einsatzzweck als untauglich erwiesen haben, wird deutlich, dass es sich anbietet direkt auf die flexible Vergabe mehrerer Rechtegruppen an Nutzer zurückzugreifen (Abschnitt 4.5), da diese keine offenkundigen Nachteile besitzt. Dieses System ist zwar technisch am aufwändigsten umzusetzen und möglicherweise für den Administrator auch am schwierigsten zu erlernen, dafür ist die Pflege der Zugriffsrechte langfristig am übersichtlichsten, sofern sinnvolle Bezeichnungen für die Benutzergruppen gewählt und die einzelnen Zugriffsrechte logisch gruppiert an diese Gruppen vergeben werden („Trainer“, „Kundenbetreuung“, „Add-on: Werbefrei“, „Add-on: Einzelsitzung“). Nachfolgend soll daher die technische Seite der Rechteprüfung, unter Verwendung dieses Verfahrens, näher beleuchtet werden.

4.7. Umsetzung der gewählten Strategie

Die notwendigen Änderungen an der SQL-Datenbank folgen direkt aus der textuellen Beschreibung: Es wird jeweils eine Tabelle für Rechte (`Permission`) und Rechtegruppen (`Permissiongroup`) angelegt. Dazu kommen noch zwei Relationstabellen, um die Rechte mit Rechtegruppen (`PermissionPermissiongroup`) und die Rechtegruppen mit Nutzern (`PermissiongroupPerson`) zu verknüpfen (Abbildung 4.1).

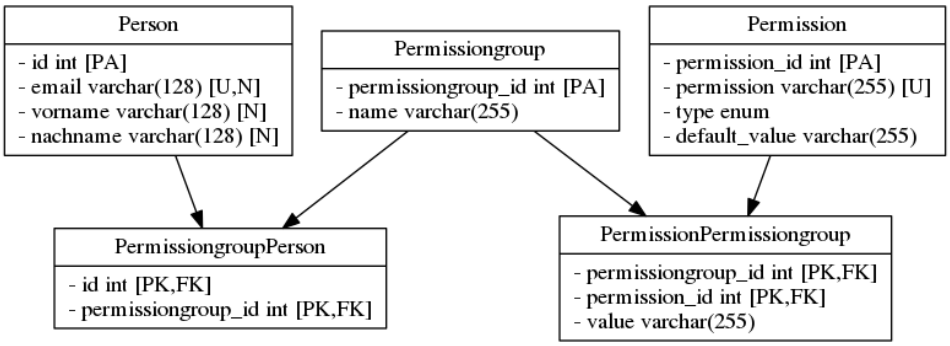


Abbildung 4.1.: Datenbankschema mit den notwendigen Änderungen für Autorisierung mittels mehrerer Rechtegruppen pro Nutzer.

Innerhalb der Anwendung werden bei der Prüfung eines Rechtes zuerst der Typ und der Basiswert (`default_value`) ermittelt. Anschließend werden die Werte aller Gruppen des Nutzers gelesen. Der Basiswert und die Werte der Gruppen werden zur Ermittlung des Resultats abhängig vom Typ des Rechts zusammengefügt. In der Beispielimplementierung existiert lediglich der Typ `boolean` für boolesche Werte. Der resultierende Wert ist `true`, wenn der Basiswert oder der Wert mindestens einer Gruppe `true` ist und sonst `false`.

Algorithmus 4.1 Zusammenführung der Einzelwerte von Zugriffsrechten des Typs boolean.

$R \leftarrow default\_value$
for all $value \in group\_values$ **do**
  $R \leftarrow R$ **or** $value$
end for
return R

Für einen fiktiven Typ `number`, beispielsweise für eine maximale Anzahl von etwas, könnte das Resultat entweder der höchste Einzelwert oder die Summe aller Einzelwerte sein. Ersteres wird in den meisten Fällen die intuitivere Wahl sein.

Zur Umsetzung der eingangs beschriebenen dynamischen Rechte wäre es denkbar die `Permission`-Tabelle um eine Spalte `is_dynamic` zu erweitern, die bei positivem Wert das Recht vor dem Administrator verbirgt. Wenn ein derartiges dynamisches Recht erzeugt wird, fügt die Anwendung dieses mit einem generischen Bezeichner wie beispielsweise `can_view_profile_123` mit „Nicht erlaubt“ als Basiswert in die Tabelle ein. `123` bezeichnet hier den Primärschlüs-

sel der Nutzertabelle. Damit ein Nutzer dieses Recht auch an andere Nutzer vergeben kann, wird zusätzlich eine Rechtegruppe angelegt werden, welche ebenfalls vor dem Administrator verborgen wird. Der Bezeichner der Gruppe ist beliebig, könnte der Einfachkeit halber identisch mit dem des Rechts sein. Dieser Gruppe wird das Recht mit dem Wert „Erlaubt“ zugewiesen. Der Nutzer könnte anschließend über eine simple Eingabemaske die Mitglieder dieser Gruppe konfigurieren und somit entscheiden, wer sein Profil aufrufen darf. Beim Aufruf eines Profils wird dann der konstante Teil des Bezeichners mit dem jeweiligen Primärschlüssel des aufgerufenen Profils konkateniert und anschließend die Funktion zur Prüfung des Rechts aufgerufen.

5. Fazit und Ausblick

Die Integration einer Mehrfaktorauthentifizierung ist mit geringem Aufwand auch in bestehende Systeme möglich und erhöht die Sicherheit von Benutzerkonten potentiell deutlich. Es ist jedoch nur mit einer Integration auf technischer Basis noch nicht getan. In Abschnitt 2.4 wurde bereits deutlich, dass auch die Mitarbeiter der Nutzerbetreuung entsprechend geschult werden müssen, damit die Sicherheit des Systems nicht ausgehebelt wird. Der menschliche Faktor ist, wie auch in der Einleitung festgestellt [Uni16], noch immer eine der größten Sicherheitslücken. Damit einhergehend muss die User Experience bei der Verwendung der Mehrfaktorauthentifierung so gestaltet sein, dass die zusätzliche Authentifizierung dem Benutzer nicht unverhältnismäßig zur Last fällt oder er zudem den Mehrwert durch die Verwendung erkennt. Andernfalls ist die Wahrscheinlichkeit hoch, dass der Benutzer die Mehrfaktorauthentifizierung wieder deaktiviert - oder gar nicht erst aktiviert. U2F ist in der Anschaffung mit Kosten verbunden, damit ist die Hürde für die erstmalige Verwendung relativ hoch. Dafür ist die spätere Nutzung sehr bequem, denn es muss nur ein Knopf an der Smartcard betätigt werden. TOTP hingegen ist für viele Menschen leicht zu nutzen, da ein großer Teil der Bevölkerung bereits ein Smartphone besitzt. Es muss aber bei jeder Authentifizierung das Smartphone genommen, entsperrt, die Anwendung gestartet und das Einmalkennwort eingegeben werden. Ein Anbieter sollte also mehrere sichere Verfahren zur Authentifizierung anbieten, um dem Nutzer verschiedene Auswahlmöglichkeiten zu geben.

Auf technischer Seite muss regelmäßig geprüft werden, ob die umgesetzten Verfahren noch sicher sind oder anderfalls mittelfristig entfernt werden. Dabei ist zu diskutieren, wie man mit Nutzern verfährt, die ausschließlich unsichere Verfahren mit ihrem Konto verknüpft haben. Auch ist die Sicherheit des Gesamtsystems nicht zu vernachlässigen. Die sicherste Mehrfaktorauthentifizierung ist nutzlos, wenn die SQL-Datenbank mit allen Nutzerdaten ohne Authentifizierung für jedermann zugänglich ist. Nachfolgend finden sich da-

her Ansätze zur weiteren Erhöhung des Sicherheitsniveaus durch zusätzliche Maßnahmen sowie zur Verbesserung und dauerhaften Pflege des bereits Umgesetzten.

5.1. Zusätzliches Härten des Systems

Abhängig von dem Wirtschaftssektor in dem sich die Anwendung befindet, kann es als Dienstanbieter sinnvoll sein noch weitere Maßnahmen zu ergreifen, um die Nutzerdaten zu schützen. Die Server einer Anwendung im Finanzsektor sollten keine virtualisierten Server bei einem Cloud-Anbieter sein, sondern sich direkt auf dem Grundstück des Betreibers befinden und von einem speziell ausgebildeten Sicherheitsdienst überwacht werden. Ebenso sollte hier auf eine Verschlüsselung der Festplatten aller beteiligten Server gesetzt werden.

Eine möglichst lückenlose Protokollierung aller Zugriffe ist auch in Bezug auf die öffentliche Wahrnehmung ein nicht zu unterschätzender Faktor. Im Falle eines Datendiebstahls ist es für betroffene Nutzer beruhigender und transparenter zu erfahren, dass die Ursache und alle betroffenen Informationen präzise ermittelt werden konnten, der Fehler behoben wurde und dadurch auch zukünftig keine Gefahr mehr darstellt.

Bevor sich jedoch Gedanken über spezialisierte Abwehrmechanismen gemacht werden, muss die Basispflege der eingesetzten Systeme sichergestellt werden. Wie auch beim heimischen Computer ist die regelmäßige Aktualisierung aller eingesetzten Software-Komponenten essentiell. Auch sollten die Entwickler geschult sein gängige Sicherheitslücken in der selbst entwickelten Anwendung zu vermeiden. Eine Liste gängiger Sicherheitslücken wird unter der Bezeichnung ‘Top 10 Project’ vom Open Web Application Security Project (OWASP) herausgegeben [Opea].

5.2. Verbesserung der Beispielimplementierung

In der Beispielimplementierung gibt es noch einige wenige unrunde Stellen, die für die produktive Verwendung verbessert werden sollten:

1. An erster Stelle steht hier die Verwendung der statisch kompilierten OpenSSL-Binary (Abschnitt 3.2.3). OpenSSL enthielt bereits des öfteren sicherheitsrelevante Programmierfehler [Opeb]. Durch das Nicht-

verwenden der durch das Betriebssystem bereitgestellten Version von OpenSSL ist man gezwungen sich selbst über die Veröffentlichung von neuen OpenSSL-Versionen zu informieren und gegebenenfalls die manuell kompilierte Version von OpenSSL zu aktualisieren. Die zentral gestellte Version wird, wenn nötig, durch den Hersteller des Betriebssystems aktualisiert und für alle Programme durch die Paketverwaltung zur Verfügung gestellt.

2. Auch das Cookie-basierte Sitzungssystem ist für die produktive Verwendung möglicherweise suboptimal: Die Anwendung hat keine Möglichkeit Daten der Sitzung mit sofortiger Wirkung zu löschen oder zu ändern. Ein Nutzer mit bösartigen Absichten könnte ältere Versionen des Cookies aufbewahren und diese für zukünftige Anfragen erneut verwenden. Im Kontext der Mehrfaktorauthentifizierung mittels U2F führt dies wohl nicht zu sicherheitsrelevanten Problemen: Durch den `counter`-Wert werden Replay-Angriffe verhindert. Es wäre aber denkbar, dass in Zukunft weitere Informationen in dem Cookie hinterlegt werden, bei denen es diesen Schutz nicht gibt. Mittelfristig sollte auf serverbasierte Sitzungen umgestellt werden. Es würde ausreichen den Ordner der PHP-Sitzungen zwischen den einzelnen Knoten zu synchronisieren oder einen Dienst wie Redis [Fav] zur Speicherung der PHP-Sitzungen zu verwenden.

3. Der Ablauf zur Aktivierung und Authentifizierung mittels des zweiten Faktors ist noch relativ umständlich und für unerfahrene Nutzer möglicherweise unverständlich. Dieser Arbeitsablauf könnte von der Benutzerfreundlichkeit her verbessert werden, beispielsweise indem die einzelnen Schritte mit Screenshots oder Beschreibungen genauer erklärt und einzelne Schritte, wenn möglich, zusammengefasst werden. Andernfalls könnte es zu der zu Beginn beschriebenen Nichtverwendung der Mehrfaktorauthentifizierung kommen.

5.3. Ausblick

Es werden regelmäßig inhärente Probleme in Verfahren zur Authentifizierung und Speicherung von Authentifizierungsdaten gefunden. Während es noch vor einigen Jahren üblich war Kennwörter mittels der MD5-Hashfunktion „sicher“

in der Datenbank abzulegen, gilt dies heutzutage nicht mehr als zeitgemäß. Zuletzt wurde im Juli 2015 im Rahmen der Password Hashing Competition eine neue Empfehlung (Argon2i) zur sicheren Speicherung von Kennwörtern ausgesprochen [Aum]. Es ist jedoch wahrscheinlich, dass auch bei Argon2i durch Kryptoanalyse Fehler im Algorithmus gefunden werden, wodurch das Ermitteln eines so gespeicherten Kennworts wirtschaftlich wird. Die Sicherheit von Verfahren zur Speicherung von Kennwörtern muss also regelmäßig im Kontext des jeweils aktuellen Stands der Technik neu geprüft und gegebenenfalls ein besseres Verfahren entwickelt werden.

Ebenfalls ein interessanter Gegenstand für weitere Forschung ist die kennwortlose Authentifizierung: Durch die Tatsache, dass Dienste kein Kennwort verlangen, kann auch kein Kennwort bei einem Angriff entwendet werden. Auch kann der Nutzer sein Kennwort nicht vergessen, weil er keines benötigt. Die Authentifizierung erfolgt beispielsweise über einen nur kurze Zeit gültigen Link, der an die hinterlegte E-Mail-Adresse gesendet wird. Verfahren, wie das mittlerweile eingestellte Mozilla Persona, [Lar14] hatten zum Ziel eine kennwortlose Authentifizierung durch Integration in den Webbrowser massentauglich zu machen.

Es wird deutlich, dass es im Bereich der Authentifizierung und Autorisierung noch Aspekte gibt, die nicht endgültig gelöst sind und es möglicherweise auch niemals werden. Diese wissenschaftliche Arbeit kann darauf aufgrund ihres eigentlichen Fokus keine Antwort geben, aber lässt sie daher für zukünftige Forschung offen.

A. Inhalt der beigelegten CD-ROM

Auf der beigelegten CD-ROM finden sich, neben einer digitalen Version dieser Arbeit, der Quelltext der in Kapitel 3 und 4.7 entwickelten Beispielimplementierung und etwaiger eingesetzter Drittbibliotheken.

Die Quelldateien sind UTF-8-kodiert und haben Unix-Zeilenenden (LF).

fitnessKOMPLEX/* Die, um Mehrfaktorauthentifizierung und Autorisierung erweiterte, Anwendung.

fitnessKOMPLEX.diff Übersicht der Änderungen, die an der Anwendung vorgenommen wurden. Dateien, die hier nicht aufgelistet sind, wurden unverändert übernommen.

fitnessKOMPLEX.tar Tar-Archiv des **fitnessKOMPLEX**-Ordners, damit symbolische Verknüpfungen und Unix-Dateirechte erhalten bleiben.

LICENSE Liste der Lizenzen eingesetzter Drittbibliotheken.

openssl/* Quelltext der mitgelieferten, statisch kompilierten OpenSSL-Binary (siehe Abschnitt 3.2.3).

U2F-Testscript/* Eigenständiges Testskript für die in Abschnitt 3.3 entwickelte PHP-Klasse zur Kommunikation mit einer U2F-Smartcard. Zur Verwendung ist ein PHP-fähiger Webserver mit TLS-Zertifikat notwendig.

B. Einrichtung von Universal Second Factor

Das U2F-Verfahren ist derzeit nur im Google Chrome (nativ) und Mozilla Firefox (Add-on) [Chm] nutzbar. Bei Verwendung einer aktuellen Version von Microsoft Windows sollte der Treiber für die Smartcard beim ersten Einstecken automatisch installiert werden. Für GNU/Linux-Systeme ist es im Regelfall erforderlich eine neue udev-Regel zu ergänzen, damit die Smartcard vom Webbrowser angesteuert werden kann:

1. Herunterladen der aktuellen Regelversion von `https://github.com/Yubico/libu2f-host/blob/master/70-u2f.rules`.
2. Speichern der Regel in `/etc/udev/rules.d`.
3. Neustart des Systems.

Zur Prüfung der Funktionsfähigkeit der Smartcard kann die U2F-Demoseite von Yubico genutzt werden:

1. Aufruf der Demo-Seite in einem kompatiblen Webbrowser: `https://demo.yubico.com/u2f`.
2. Auswahl des Reiters „Register“.
3. Eingabe von beliebigen Zugangsdaten (diese werden anschließend im Klartext angezeigt).
4. Nach Absenden des Formulars: Betätigung des Knopfes an der Smartcard.
5. Es sollte gezeigt werden, dass die Registrierung erfolgreich war.

Abkürzungen

AES	Advanced Encryption Standard
API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
CGI	Common Gateway Interface
DER	Distinguished Encoding Rules
ECDSA	Elliptic Curve Digital Signature Algorithm
ELSTER	ELektronische STeuerERklärung
FIDO	Fast IDentity Online
FIPS	Federal Information Processing Standard
HMAC	Keyed-Hash Message Authentication Code
HOTP	HMAC-Based One-Time Password Algorithm
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IRC	Internet Relay Chat
JSON	JavaScript Object Notation
MAC	Message Authentication Code
MD5	Message-Digest Algorithm 5
NIST	National Institute of Standards and Technology
OTP	One-Time Password

OWASP Open Web Application Security Project

PEM Privacy Enhanced Mail

PGP Pretty Good Privacy

PHP PHP: Hypertext Preprocessor

PIN Persönliche Identifikationsnummer

QR-Code Quick Response-Code

RBAC Role-based access control

RHEL Red Hat Enterprise Linux

SHA Secure Hash Algorithm

SIM Subscriber Identity Module

SMS Short Message Service

TAN Transaktionsnummer

TLS Transport Layer Security

TOTP Time-based One-time Password Algorithm

U2F Universal Second Factor

URL Uniform Resource Locator

Abbildungsverzeichnis

Algorithmenverzeichnis

Listingverzeichnis

Literatur

[129] 129.69.226.230. *Operator*. URL: http://minecraft-de.gamepedia.com/index.php?title=Operator&diff=273711&oldid=216053 (besucht am 24. 11. 2016).

[Arc16] Scott Arciszewski. *Updated constraint on OpenSSL*. Okt. 2016. URL: https://github.com/defuse/php-encryption/pull/309#issuecomment-253472651 (besucht am 17. 11. 2016).

[Aum] Jean-Philippe Aumasson. *Password Hashing Competition*. URL: https://password-hashing.net/ (besucht am 05. 12. 2016).

[BBL15] Dirk Balfanz, Arnar Birgisson und Juan Lang. *FIDO U2F Javascript API*. Mai 2015. URL: https://fidoalliance.org/specs/fido-u2f-javascript-api-ps-20150514.pdf.

[BE15] Dirk Balfanz und Jakob Ehrensvard. *FIDO U2F Raw Message Formats*. Mai 2015. URL: https://fidoalliance.org/specs/fido-u2f-raw-message-formats-ps-20150514.pdf.

[BH15] Dirk Balfanz und Brad Hill. *FIDO AppID and Facet Specification*. Mai 2015. URL: https://fidoalliance.org/specs/fido-appid-and-facets-ps-20150514.pdf.

[BL14] Daniel J. Bernstein und Tanja Lange. *SafeCurves: Introduction*. Jan. 2014. URL: https://safecurves.cr.yp.to/ (besucht am 12. 11. 2016).

[Chm] Paweł Chmielowski. *U2F Support Add-On*. URL: https://addons.mozilla.org/en-GB/firefox/addon/u2f-support-add-on/ (besucht am 11. 11. 2016).

[Eye15] EyeLock Inc. *EyeLock's myris Is First and Only Iris Authenticator for New FIDO Open Industry Standard*. Jan. 2015. URL: http://www.prnewswire.com/news-releases/eyelocks-myris-is-first-and-only-iris-authenticator-for-new-fido-open-industry-standard-300015710.html (besucht am 11. 11. 2016).

[Fav] Nicolas Favre-Felix. *PHP Session handler*. URL: https://github.com/phpredis/phpredis#php-session-handler (besucht am 27. 11. 2016).

[Fed] Federal Financial Institutions Examination Council. *Authentication in an Internet Banking Environment*. URL: https://www.ffiec.gov/pdf/authentication_guidance.pdf.

[FID14] FIDO Alliance. *FIDO 1.0 Specifications are Published and Final Preparing for Broad Industry Adoption of Strong Authentication in 2015*. Dez. 2014. URL: https://fidoalliance.org/fido-1.0-specifications-published-and-final/ (besucht am 11. 11. 2016).

[FK92] David F. Ferraiolo und D. Richard Kuhn. *Role-Based Access Controls*. Okt. 1992. URL: https://web.archive.org/web/20160303234840/http://csrc.nist.gov/groups/SNS/rbac/documents/ferraiolo-kuhn-92.pdf (besucht am 24. 11. 2016).

[Git16] GitHub, Inc. *Email replies disclose "mute the thread" token*. Sep. 2016. URL: https://bounty.github.com/researchers/h8rry.html (besucht am 11. 11. 2016).

[Goo] Google Inc. *Google Authenticator*. URL: https://play.google.com/store/apps/details?id=com.google.android.apps.authenticator2.

[Goo14] Google Inc. *u2f-api.js*. 2014. URL: https://github.com/google/u2f-ref-code/blob/80a30a38178a5a277c4cc13df36c82671d85d881/u2f-gae-demo/war/js/u2f-api.js (besucht am 17. 11. 2016).

[Goo15] Google Inc. *Key Uri Format*. 2015. URL: https://github.com/google/google-authenticator/wiki/Key-Uri-Format (besucht am 11. 11. 2016).

[HIBP] *';–have i been pwned?* URL: https://haveibeenpwned.com/ (besucht am 11. 11. 2016).

[Hor] Taylor Hornby. *php-encryption*. URL: https://github.com/defuse/php-encryption/tree/v1.2.1 (besucht am 14. 11. 2016).

[Lar14] Frederic Lardinois. *Mozilla Stops Developing Its Persona Sign-In System Due To Low Adoption*. März 2014. URL: https://techcrunch.com/2014/03/08/mozilla-stops-developing-its-persona-sign-in-system-because-of-low-adoption/ (besucht am 05. 12. 2016).

[Mar] Devin Martin. *KeeOtp*. URL: http://keepass.info/plugins.html#keeotp (besucht am 11. 11. 2016).

[MI16] Yusuf Motara und Barry Irwin. "SHA-1 and the Strict Avalanche Criterion". In: *CoRR* abs/1609.00616 (2016). URL: http://arxiv.org/abs/1609.00616.

[Nat13] National Institute of Standards and Technology. *Digital Signature Standard (DSS)*. Juli 2013. URL: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf (besucht am 09. 11. 2016).

[Opea] Open Web Application Security Project. *Category:OWASP Top Ten Project*. URL: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project (besucht am 03. 12. 2016).

[Opeb] OpenSSL Software Foundation. *Vulnerabilities*. URL: https://www.openssl.org/news/vulnerabilities.html (besucht am 27. 11. 2016).

[RFC2104] Hugo Krawczyk, Mihir Bellare und Ran Canetti. *HMAC: Keyed-Hashing for Message Authentication*. RFC 2104. RFC Editor, Feb. 1997. URL: http://www.rfc-editor.org/rfc/rfc2104.txt.

[RFC2617] John Franks u. a. *HTTP Authentication: Basic and Digest Access Authentication*. RFC 2617. RFC Editor, Juni 1999. URL: http://www.rfc-editor.org/rfc/rfc2617.txt.

[RFC3174] D. Eastlake und P. Jones. *US Secure Hash Algorithm 1 (SHA1)*. RFC 3174. RFC Editor, Sep. 2001. URL: http://www.rfc-editor.org/rfc/rfc3174.txt.

[RFC4226] D. M'Raihi u. a. *HOTP: An HMAC-Based One-Time Password Algorithm*. RFC 4226. RFC Editor, Dez. 2005. URL: http://www.rfc-editor.org/rfc/rfc4226.txt.

[RFC4648] S. Josefsson. *The Base16, Base32, and Base64 Data Encodings*. RFC 4648. RFC Editor, Okt. 2006. URL: http://www.rfc-editor.org/rfc/rfc4648.txt.

[RFC5280] D. Cooper u. a. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280. RFC Editor, Mai 2008. URL: http://www.rfc-editor.org/rfc/rfc5280.txt.

[RFC5480] S. Turner u. a. *Elliptic Curve Cryptography Subject Public Key Information*. RFC 5480. RFC Editor, März 2009. URL: http://www.rfc-editor.org/rfc/rfc5480.txt.

[RFC6238] D. M'Raihi u. a. *TOTP: Time-Based One-Time Password Algorithm*. RFC 6238. RFC Editor, Mai 2011. URL: http://www.rfc-editor.org/rfc/rfc6238.txt.

[RFC6265] A. Barth. *HTTP State Management Mechanism*. RFC 6265. RFC Editor, Apr. 2011. URL: http://www.rfc-editor.org/rfc/rfc6265.txt.

[San] Salvatore Sanfilippo. *Redis Security*. URL: http://redis.io/topics/security (besucht am 24. 11. 2016).

[sec] secunet Security Networks AG. *Sicherheitsstick für ELSTER*. URL: https://www.sicherheitsstick.de/ (besucht am 11. 11. 2016).

[Sri+15] Sampath Srinivas u. a. *Universal 2nd Factor (U2F) Overview*. Mai 2015. URL: https://fidoalliance.org/specs/fido-u2f-overview-ps-20150514.pdf.

[The] The PHP Group. *Session Handling*. URL: http://php.net/manual/en/book.session.php (besucht am 14. 11. 2016).

[Uni16] University of Luxembourg. *Social engineering: password in exchange for chocolate*. Mai 2016. URL: http://wwwen.uni.lu/university/news/latest_news/social_engineering_password_in_exchange_for_chocolate (besucht am 11. 11. 2016).

[X.680] International Telecommunication Union. *Information Technology — Abstract Syntax Notation One (ASN.1): Specification of Basic Notation*. ITU-T Recommendation X.680. Juli 2002. URL: http://www.itu.int/ITU-T/studygroups/com17/languages/X.680-0207.pdf.

[X.690] International Telecommunication Union. *Information Technology — ASN.1 Encoding Rules — Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER)*. ITU-T Recommendation X.690. Juli 2002. URL: http://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf.

[Yub] Yubico AB. *U2F — FIDO Universal 2nd Factor Authentication*. URL: https://www.yubico.com/about/background/fido/ (besucht am 11. 11. 2016).

Eidesstattliche Erklärung

Hiermit versichere ich, dass die vorliegende Arbeit über secureFIT - Mehrfaktor-Authentifizierung und Autorisierung selbstständig verfasst worden ist, dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt worden sind und dass die Stellen der Arbeit, die anderen Werken – auch elektronischen Medien – dem Wortlaut oder Sinn nach entnommen wurden, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind.

Tim Düsterhus, Sassenberg, 12. Dezember 2016

Ich erkläre mich mit einem Abgleich der Arbeit mit anderen Texten zwecks Auffindung von Übereinstimmungen sowie mit einer zu diesem Zweck vorzunehmenden Speicherung der Arbeit in eine Datenbank einverstanden.

Tim Düsterhus, Sassenberg, 12. Dezember 2016